Creating a GUI for GitConfig

MacTech Magazine
June • 2010

MACTECH®

The Journal of Apple Technology

Is the iPad an Enterprise Tool?

bash String Operations

CORESEC

Security for Admins and Programmers

MACTECH.COM

$8.95 US, $12.95 Canada

06

0 32128 74887 8

ISSN 1067-8360 Printed in U.S.A.

Business ~~Weather~~ forecast.

HansaWorld USA, Inc. 1501 Broadway #12067 & 12068, 12th Floor, New York, NY 10036-5601, USA. Phone: +1 646-459-0603, Email: usa@hansaworld.com

NEW

Switch to Mac
has never been easier.

Parallels Desktop ® Switch to Mac Edition

Ready. Set. Switch.

- Complete Moving Suite
- Interactive Video Tutorials
- Step-by-Step Easy Migration

- Run Windows and Mac Side-by-Side without Rebooting
- Enjoy Your Favorite USB Devices
- Plus $175 Bonus Windows Software

Parallels Desktop Switch to Mac Edition lets you move programs, documents, media and more right from your PC to your new Mac. Then enjoy the best of both worlds and run Windows and Mac OS X side by side.

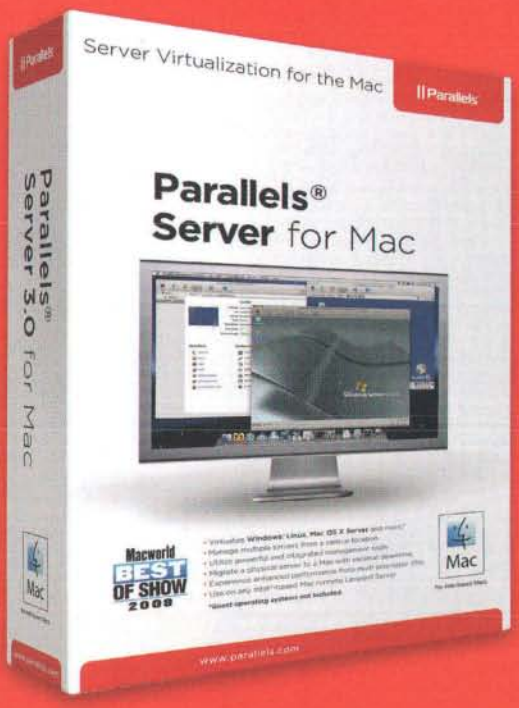
Learn more at
www.parallels.com/products/desktop/stm

Parallels
Optimized Computing

Phone +1 (425) 282-6448

Parallels™

Parallels® Server for Mac



Make your Mac Server go farther.

Run any OS you choose.

Run any application on the Apple Xserve today.

The world's first server virtualization solution for the Mac platform.

Reduce Server Count by 84%

Save 60% on IT Budget

Oregon City School District

Performance

Full scale hypervisor solution with bare metal architecture.

Scalability

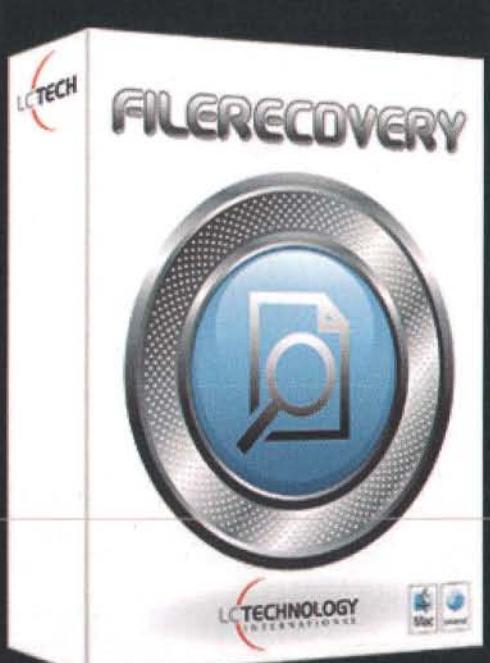
Virtualize Mac OS X Server, Windows, Linux and more with support for 32- and 64-bit platform and guest OS support.

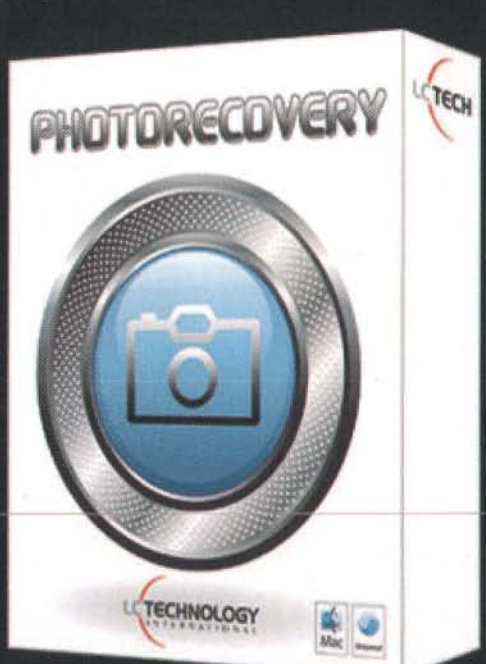
Flexibility

Hardware-ready for seamless integration into existing IT infrastructures with built-in VM management and maintenance tools.

Buy now at www.parallels.com or email offer@parallels.com

DATA RECOVERY FOR YOUR MAC





WWW. LC-TECH.COM 1-866-603-2195

LC TECHNOLOGY INTERNATIONAL

Mac

TABLE OF CONTENTS

From the Editor

As this issue is going to press, MacTech just announced the very first MacTech Technical Conference. Set to take place November 3-5 in Los Angeles, California, this is going to be a special few days. We're just about to announce speakers and general line up, which should be done once you receive this issue in the mail. If you would like to share your knowledge and solutions with your peers, please submit a speaker application (http://www.mactech.com/conference/speakers-app). As a speaker or attendee, we hope you'll clear your calendar and join us for this amazing event.

The idea of the show is to present topics that will be new to you, challenge you and inspire you. There will be many opportunities to meet your peers and exchange ideas. The MacTech Conference will provide learning beyond the technology. Keep an eye on http://www.mactech.com/conference and @mactechconf on Twitter.

This month, our cover story features a new monthly column: CoreSec. All aspects of computer security are only becoming more relevant to all computer users, both professional and recreational. Mac OS X is not excluded from the many security issues that face us. Mike Hjörleifsson brings his expertise to guide us through the minefield of terms, options and exploits.

We all love the new shiny from Apple. However, what happens when, as a company, Apple does something you may not agree with? Of if it seems like Apple is "getting too big?" Bloggers have been accusing Apple of becoming like Microsoft, but Apple fans see it differently. Of course, the reality is somewhere in-between. Michael Swaine taps into this trend with, "Oh Noes! Apple is Turning into the Borg!"

José Cruz continues his article on writing a GUI front-end for the command-line git-config application for the Git version control program. Whether you use git-config or not, it's a great example of wrapping a command-line tool in a GUI that's a little easier to use.

This month's Mac in the Shell presents some very often-overlooked bash string operators. For those that use the built-in operators, calls to external tools can be dropped, and performance can increase for everyone on the system. Check out how in, "bash String Operations."

In MacEnterprise, Greg Neagle examines the viability of using the iPad as an Enterprise device. His analysis is in "iPad in the Enterprise." His findings may corroborate your day to day work as more and more Executives purchase iPads for their use.

José Cruz this month also brings us a tutorial on how to teach BBEdit syntax-aware editing for different languages. Support for C is built-in. What happens, though, when you need Python, Ruby or other language support?

Finally, this month's MacTech Spotlight highlights Boisy Pitre. Boisy is the owner of Tee Boy, located in Opelousas, Louisiana, and quite an interesting person. Check out his approach to Mac development in this month's MacTech Spotlight.

Ed Marczak,
Executive Editor

Communicate With Us

Department E-Mails

Orders, Circulation, & Customer Service
cust_service@mactech.com

Press Releases
press_releases@mactech.com

Ad Sales
adsales@mactech.com

Editorial
editorial@mactech.com
(Authors only, no pr)

Accounting
accounting@mactech.com

Marketing
marketing@mactech.com

General
info@mactech.com

Web Site
http://www.mactech.com

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact MacTech Magazine Customer Service at 877-MACTECH

We love to hear from you! Please feel free to contact us with any suggestions or questions at any time.

Write to letters@mactech.com or editorial@mactech.com as appropriate.

MACTECH®
The Journal of Macintosh Technology

A publication of XPLAIN CORPORATION

Canada Post: Publications Mail Agreement #41513541
Canada Returns to be sent to: Bleuchip International, P.O. Box 25542, London, ON N6C 6B2

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 705 Lakefield Road, Suite I, Westlake Village, CA 91361. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

SWAINE MANOR

Oh Noes! Apple is Turning into the Borg!

...Or IBM. Or Google. Or Something. Anyway, It's Time to Panic.

by Michael Swaine

Blame It on Rio

I blame the Brazilians.

I was shrugging off all this "Apple is turning into Microsoft" hysteria until I got the call from Guilherme de Camargo Barros of *Galiliu Magazine* and was forced to really think about the question. Guilherme wanted to interview me about Apple and Steve Jobs, and several of his questions were of the "has Apple become Microsoft?" variety, triggered by the report that Apple had passed Microsoft in market valuation.

If "passed" is the word I'm seeking. Do you "pass" another car when you're heading north and you see it zip by in the southbound lane? But I digress.

Being confronted with the question in an interview forced me to take it more seriously than I had been. Well, is Apple turning into Microsoft, or IBM, for that matter, or some other relic of the past? Given some of Apple's recent actions, it was a fair question.

Only the more I thought about it, the more sure I was that it was really several questions. If Apple was in fact undergoing some ghastly transformation into a hideous monster out of the Technozoic, I realized that there were several candidate monsters.

The Microsoft Meme

Apple turning into Microsoft? I kinda get it. Remember "I'd love to offer my app on the Mac but the Windows market is twenty times bigger and the incremental income wouldn't justify the cost of porting"? Now put WebOS in place of the Mac and iPhone/iPad in place of Windows. OK.

There must be some validity to the idea; it's a topic (http://bit.ly/dzca3G) on Apple's own support discussion board.

The premise seems to be that Apple's growth and its recent behavior in tromping on users and developers, as well as its *de facto* monopoly in some new market segments, are evidence that Apple is turning into Microsoft. But other conclusions could be drawn from the same evidence.

If I were forced to say what it is that Apple is turning into, I'd have to say it's turning into Apple. A bigger, stronger, more Apple-like Apple.

Take the legal harassment stuff. That could be evidence that Steve Jobs is turning into Darl McBride. (Darl who, you ask? CEO of SCO, claimed ownership of Unix, built a business model on suing everybody. At least that's how I remember it. I could be wrong. Don't sue me, Darl). The *de facto* monopoly thing? Evidence that Apple is turning into Google. The whole police state kicking-down-doors thing? Apple is turning into Big Brother. Why wasn't 1984 like *1984*? So 2010 could be.

Other Monsters

I think you could make a stronger argument that Apple is turning into Sony. It's hardly an original idea. Starting with dropping "Computer" from its name, Apple has been pretty open about its movement into a post-PC era of scads of different yet-to-be-invented consumer devices, and Steve Jobs has always admired Sony.

Or maybe Apple really is turning into Google. John Battelle says they're developing a search engine for apps. Why stop there? The iPhone OS is proof that Apple understands something that Microsoft never has: most users don't bother to put their documents in nested folders, they just dump them in one folder or on the desktop and search

REMOTE ACCESS FOR YOUR MAC.

HOW REMOTE S UP TO YOU.

Get your Mac ready for summer by setting up free remote access. Then all you need is an Internet connection to log in and control your desktop, all your files and apps – anytime, anywhere.

LogMeIn

Free

for what they want. Google understands that search is a key operating system function; maybe Apple does, too.

I'm unconvinced. I had dealings with Apple and Steve Jobs back in the 1980s, and I don't remember this benign company that is implied by these laments over Apple turning into something different. If I were forced to say what it is that Apple is turning into, I'd have to say it's turning into Apple. A bigger, stronger, more Apple-like Apple. It's acting pretty much like it's always acted: arrogant, controlling, and bold. Only now, it has more power.

And here's one consequence of Apple's embrace of post-PC products that I'm not seeing mentioned in all these articles: you can no longer turn your Apple products into aquariums and hamster cages. Now that's a shame.

About The Author

Michael Swaine is the former editor-in-chief of Dr. Dobb's Journal (http//www.ddj.com) and current editor of PragPub: (http//www.pragprog.com/magazine), the electronic magazine for pragmatic programmers. You can reach him at mike@swaine.com.

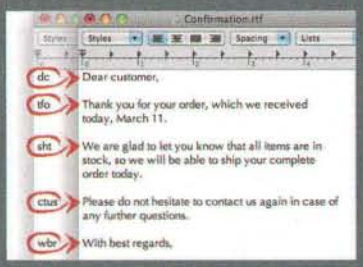
Work less. And finish faster.*

Let us do the tedious work for you.

Typing the same text over and over?

Typinator

types frequently used text for you and auto-corrects your typos.



Want to drive your Mac like a Pro?

KeyCue

helps you speed up your daily tasks with keyboard shortcuts.

Need special characters in your document?

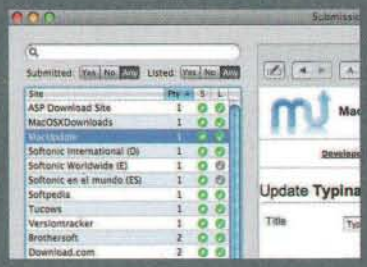
PopChar

inserts any special character into your document with just two clicks.

Have great software to sell?

Shareware Publisher

submits your software products to leading download sites.



www.ergonis.com/mactech

* **Caution:** Usage of Ergonis productivity boosters will revolutionize the way you use your Mac and make you more productive. You will have to look for another hobby!

GitConfig GUI

Building a Gui for git-config

by José R.C. Cruz

Introduction

In today's article, we follow up on last month's article, learning how to configure the Git tool to suit our revision control needs. We looked at the `git-config` command and its settings files. This month, we give the command a simple graphical interface using Xcode and AppleScript Studio.

Readers are expected to know their way around bash and AppleScript Studio. The Xcode project featured here is available from the MacTech ftp site at ftp://ftp.mactech.com.

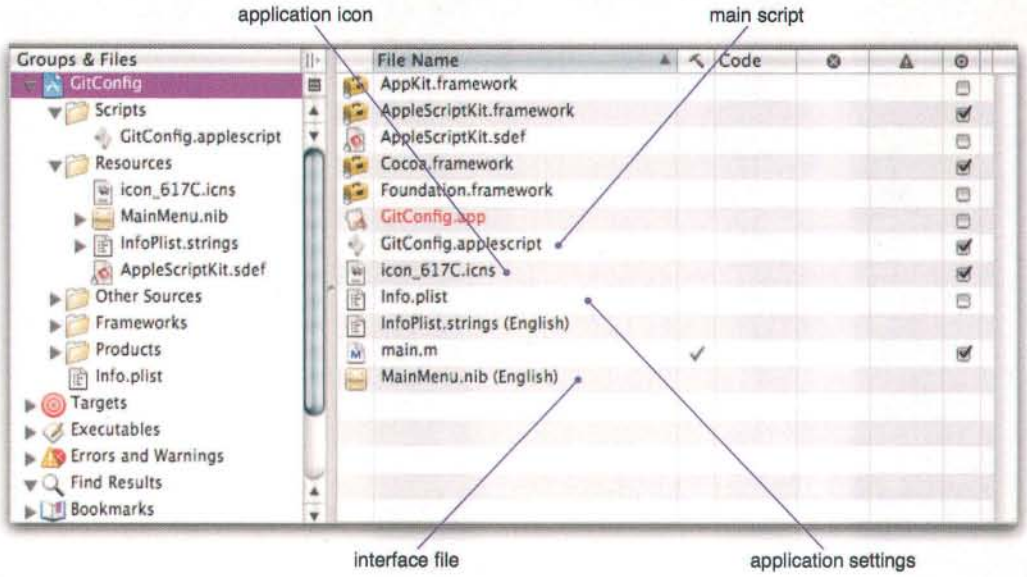


Figure 1. The GitConfig project.

Giving Config a Face

Like most Git commands, we need a shell session in order to use `git-config`. But that does not mean we cannot wrap `git-config` with a simple graphical tool. This article shows one such tool built using Xcode and AppleScript Studio. The tool will access certain group of settings and allow users make their changes easily. It will also allow users reverse their changes easily.

For reasons of length, the tool handles only a small subset of `git-config` settings. Plus, it uses only the settings file from the project repository. Readers are welcome to download the Xcode project and extend it for their own needs. Once again, the entire project is available from the MacTech ftp site at ftp://ftp.mactech.com

The project bundle

Figure 1 shows the files of the demo project, **GitConfig**. It is based on the Xcode template **AppleScript Application**, which is available from the **Project Assistant** dialog. The template already has the right files and settings needed to compile an AppleScript binary. The only files we need to attend are `MainMenu.nib`, `GitConfig.applescript`, and `Info.plist`.

Listing 1 shows the contents of the `Info.plist` file, edited for brevity. This file holds the metadata that describes the tool. The key `CFBundleIconFile` points to the graphics file that has the tool icon. The key `CFBundleIdentifier` sets the tool's unique ID. And the key `CFBundleSignature` sets the tool's four-byte signature.

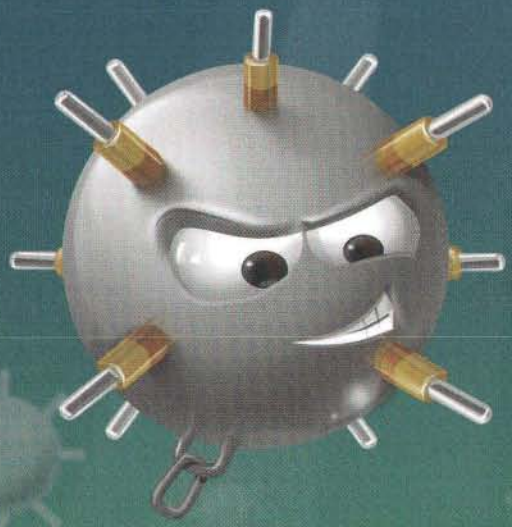
Listing 1. The Info.plist file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST
1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <!- truncated for length... ->
    <key>CFBundleIconFile</key>
    <string>icon_617C.icns</string>
    <key>CFBundleIdentifier</key>
    <string>com.mactech.anarakis.demo.git.config</string>
    <key>CFBundleInfoDictionaryVersion</key>
```

ProteMac

www.protemac.com

Protemac NetMine



Secure
your network

Protects against all types of malware, visualizes network activity, keeps traffic details.

Mac Universal

Protemac Meter

Monitor
your network

Controls network usage, traffic and safety, measures network traffic, provides statistics.

ActyMac

ActyMac
Duty Watch

Monitors your employees' monitors.

Mac Universal

www.actymac.com

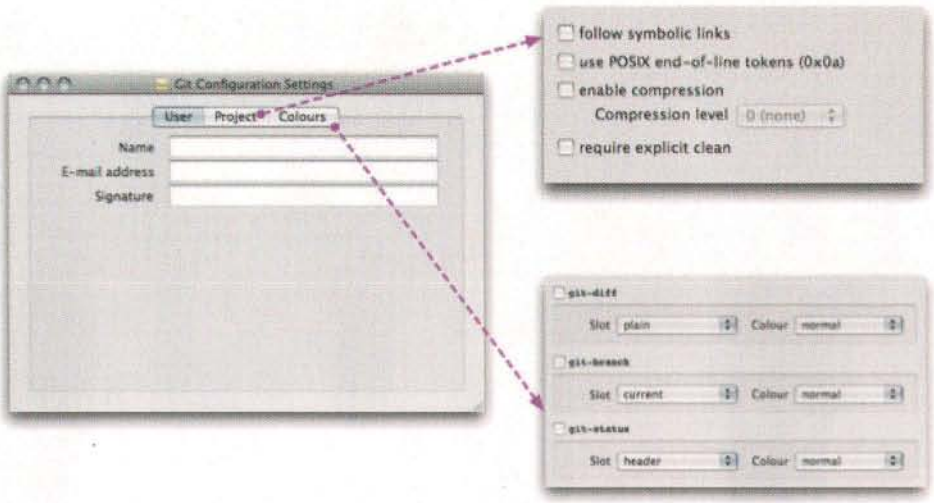


Figure 2. User interface of the GitConfig tool.

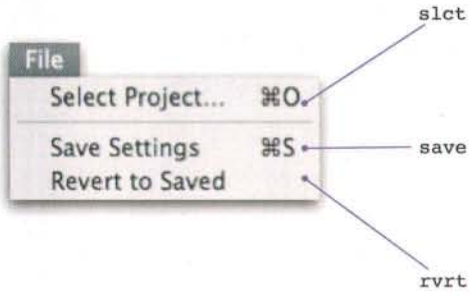


Figure 3. The GitConfig File menu.

```
    <string>6.0</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0</string>
    <key>CFBundleSignature</key>
    <string>617C</string>
    <!- truncated for length... ->
</dict>
</plist>
```

The user interface

Figure 2 shows the layout for the `MainMenu.nib` interface file. A TabView widget divides the window into three panel views. Clicking the tab label displays the panel associated to that label.

The first panel, User, has three Text Field widgets. It handles those settings that are user-specific. It is also the default panel displayed by the GitConfig tool. The second panel, Project, handles the settings that control the project repository. On it are four Check Box and one Pop Up Button widgets. Three of the checkboxes enable a specific repository action. One checkbox enables the popup widget, which gives a list of compression factors.

The third panel view, Colors, handles those settings that affect report colors. It uses three Check Box widgets to enable a color setting, as well as the underlying Box view. In each box view are two Pop Up Button widgets—the left widget gives a list of available report slots, the right widget a list of available colors.

As for the menu bar, only the File menu gets changed for the GitConfig tool. That menu gets three menu items as showed in Figure 3. The Select Project... item starts an Open Folder session wherein users can select a project directory. The Save Settings item commits all changes back to the repository's settings file. The Revert to Saved item discards those same changes and reloads the settings contained by the repository file.

The File menu also gives each menu item with a unique four-character signature. These signatures will help identify which menu item was selected by the user.

Loading the settings

The GitConfig tool asks for the location of the project repository in two ways. It can ask for the location while being launched. Or it asks for the location after the user chooses Select Project... from the File menu. Listing 2 describes the code for the event handler `will-finish-launching`. The handler begins by setting four global variables to their initial values. Then it calls the `chooseProject()` routine and stores the returned path into the `gPth` global. If the path is valid, the handler calls three separate routines, each one loading a subset of settings. It passes the `gPth` global to each routine as input.

Listing 2. Loading during launch.

```
global gPth, gUsr, gPrj, gHue

on will finish launching anApp
    — initialize the following globals
    set gPth to ""
    set gUsr to {}
    set gPrj to {}
    set gHue to {}

    — ask the user for a project repository
    set gPth to chooseProject()
    if (gPth is false) then
        display dialog "This is not a Git-controlled
directory"
    else
        — read the following repository settings
        — repository:settings:user
        set gUsr to loadGitUser from gPth

        — repository:settings:project
        set gPrj to loadGitProject from gPth

        — repository:settings:colors
        set gHue to loadGitColors from gPth
```

From print to web in a Flash. Work your magic with QuarkXPress® 8

Quark

So you're a master of print design? Time to step it up a notch. Use your existing QuarkXPress skills to design for the Web and bring your creations to full interactive glory – without having to learn Flash or coding. The intuitive design interface of QuarkXPress 8 opens a world of new possibilities. Increase your productivity and offer your clients more (both print and Web), right out of the box.

But, there's more to this box of tricks than meets the eye. Buy or upgrade to QuarkXPress 8 today to access $750+ of Exclusive Flash Resources for **FREE** to make your designs shine:

- Hundreds of exclusive, fully editable Flash assets
- Web templates, animations and video players
- Flash tutorials, eSeminars and educational resources

Purchase QuarkXPress® 8 today and access $750+ worth of Exclusive Flash Resources for **FREE**

Upgrade NOW for just $299 (before tax) or BUY for $799 (before tax)

Unleash the magic of Flash in QuarkXPress 8 visit www.quark.com/magic

```
    end if — (tPth is false)
end will finish launching — anApp
```

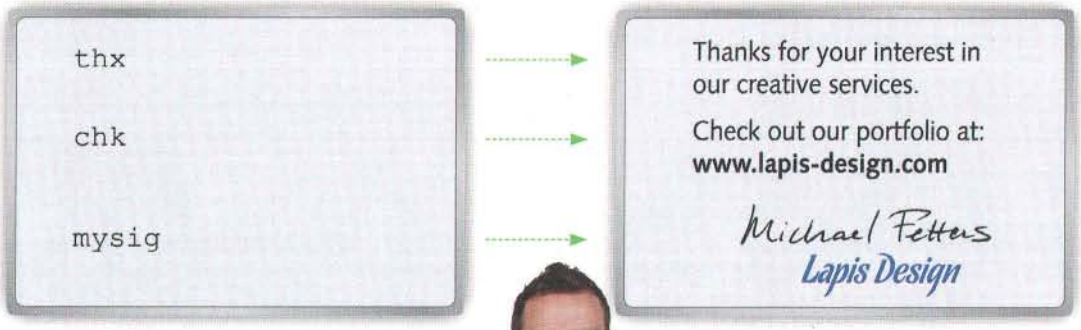
Listing 3 shows the code for the `choose-menu-item` handler. This handler begins by identifying the source of the event. If the event came from a menu item, the handler checks the signature for that item. A signature of `'slct'` means the chosen menu item is Select Project.... Knowing this, the handler runs the `chooseProject()` routine to retrieve the new project directory. It updates the `gPth` global with the results and calls the same three routines for loading the project settings. But this time, the handler checks which panel view is active on the GitConfig window. Once it identifies the panel, the handler runs the correct display routine for that panel.

Listing 3. Handling the Select Project... menu item.

```
on choose menu item aMnu
    local tNom, tTyp, tSet, tPth
    local tVal, tChg

    — initialize the following locals
    set tNom to name of aMnu as string
    set tTyp to class of aMnu as string

    — identify the calling widget
    if (tTyp is "menu item") then
        — menu:file
        if (tNom is "slct") then
            — file:repository:settings:select
            — ask the user for a project repository
            set tPth to chooseProject()
            if (tPth is false) then
                display dialog "This is not a Git-controlled
directory"
            else
                — update the following global variable
                set gPth to tPth

                — read the following repository settings
                — repository:settings:user
                set gUsr to loadGitUser from gPth

                — repository:settings:project
                set gPrj to loadGitProject from gPth

                — repository:settings:colors
                set gHue to loadGitColors from gPth

                — identify the active panel
                if (gPnl is "tUsr") then
                    — panel:settings:user
                    showGitUser from gUsr
                else if (gPnl is "tPrj") then
                    — panel:settings:project
                    showGitProject from gPrj
                else if (gPnl is "tCol") then
                    — panel:settings:colors
                    showGitColors from gHue
                end if — (gPnl is "tUsr")
            end if — (tPth is false)

            — clear the window changed state
            set document edited of gWin to false
        else if (tNom is "save") then
            — ...truncated for length

        else if (tNom is "rvrt") then
            — ...truncated for length

        end if — (tNom is "slct")

    else if (tTyp is "popup button") then
        — ...truncated for length

    end if — (tTyp is "menu item")
end choose menu item — aMnu
```

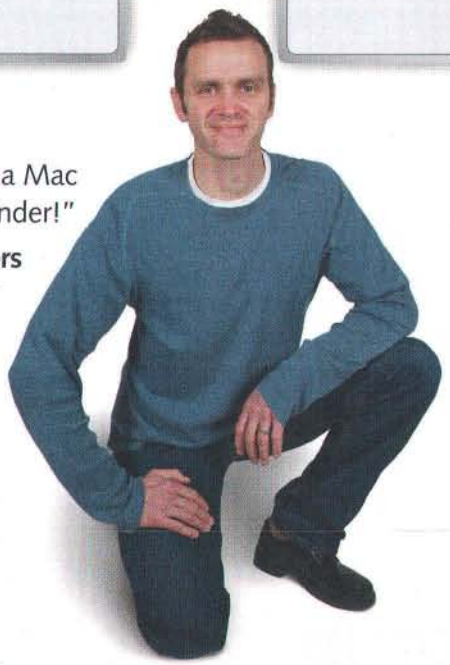
Type Less. Create More.

Effortlessly insert frequently-used words, phrases and graphics, by typing short abbreviations. Great for email, blogging and more!



"I can't work on a Mac without TextExpander!"

Michael Fetters
Mac user



textexpander

Now Available!
TextExpander touch™ for iPad, iPhone & iPod touch

Integrate TextExpander touch in your apps with our free SDK:
www.smileonmymac.com/sdk

Smile
on my mac

disclabel PDFpen pagesender textexpander

In Listing 4 is the routine `chooseProject()`. This routine begins with the display of an Open Folder dialog using the `choose-folder` OSAX function. After a user chooses the desired project directory with the dialog, the routine checks if the directory has a hidden `.git` directory. IF the check proves false, then the routine returns a `FALSE`. Otherwise, the routine converts the directory path into a POSIX string and returns the result.

Listing 4. Choosing a project directory.

```
to chooseProject()
    — prompt for a project directory
    set tPth to choose folder ¬
        with prompt text "Select a project directory"

    — check if the directory has a hidden '.git' directory
    set tLst to list folder tPth
    if (tLst contains ".git") then
        — convert the path to POSIX
        set tPth to POSIX path of tPth

        — return the conversion result
        return (tPth)
    else
        — retrieval has failed
        return (false)
    end if — (tLst contains ".git")
end chooseProject
```

In Listing 5 is the first loading routine `loadGitUser()`. This routine takes one input argument `aPth`, which holds the path to the project repository. It starts with a record of null values. Then it calls the `readConfig()` routine, passing for input the repository path and the settings key. The `readConfig()` routine returns a string value, which `loadGitUser()` stores under the right record key.

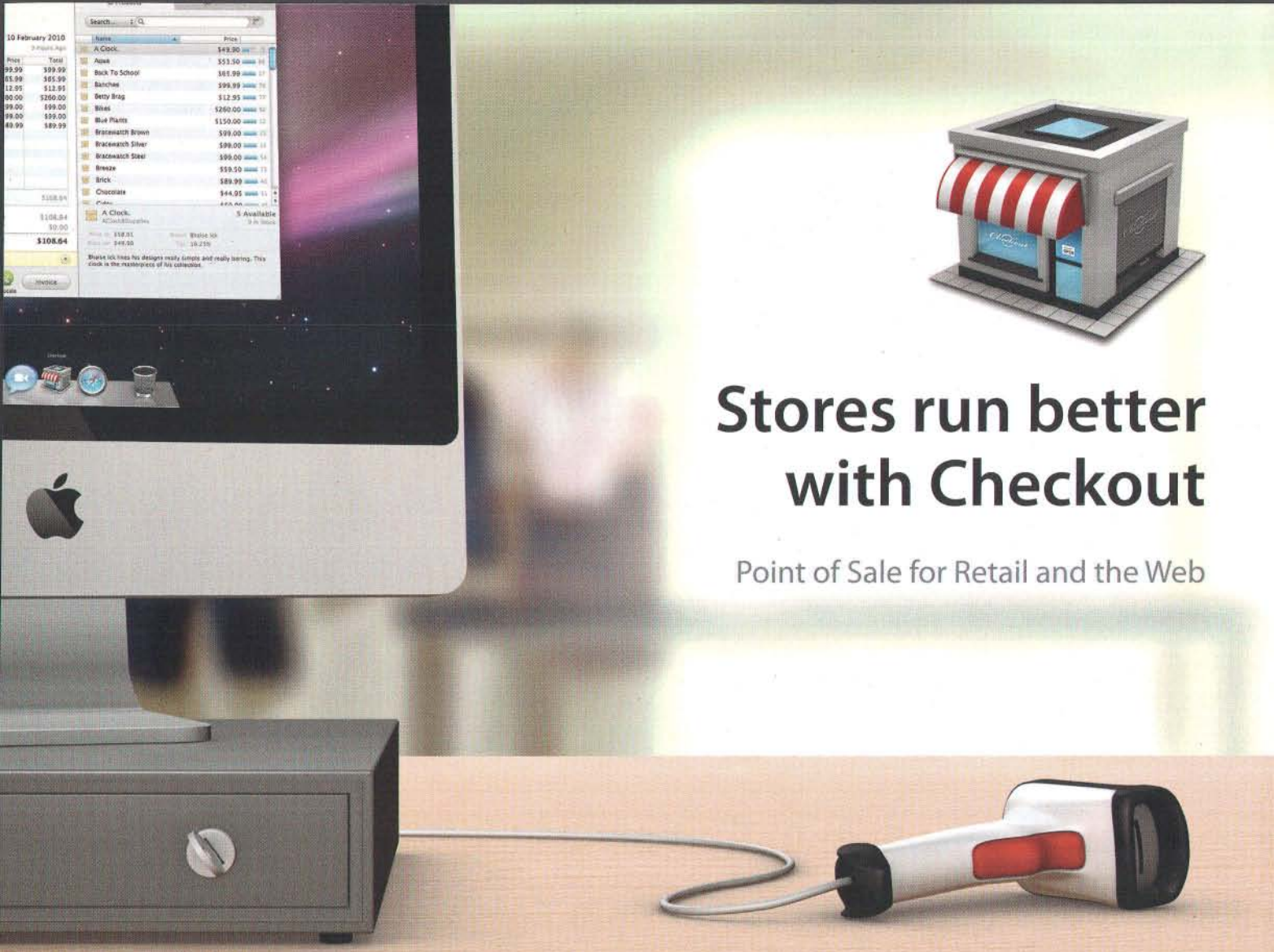
Listing 5. Reading the user settings.

```
to loadGitUser from aPth
    local tGit, tVal

    — initialize the settings list
    set tGit to {uNom:"", uEml:"", uSig:""}

    — read the following user settings
    set tVal to readConfig at aPth for "user.name"
    set uNom of tGit to tVal

    set tVal to readConfig at aPth for "user.email"
    set uEml of tGit to tVal

    set tVal to readConfig at aPth for "user.signingkey"
    set uSig of tGit to tVal

    — return the retrieval results
    return (tGit)
end loadGitUser — from aPth
```

The code behind the `readConfig()` routine is shown in Listing 6. Here, the routine prepares the command statement needed to query a settings key. It then executes the statement using the `do-shell-script` OSAX function. If no error occurred, the routine gets a string value in return. Otherwise, it assumes an error string of `'n/a'`.

Listing 6. Reading a settings key.

```
to readConfig at aPth for aKey
    local tCmd, tVal
```



Stores run better with Checkout

Point of Sale for Retail and the Web

We all know everything just works better on a Mac®. Now point of sale does too. Do your favorite retailer a favor, tell them to try Checkout for free today.

Starting from **$399**, Complete **hardware bundles** available*

Visit **www.checkoutapp.com** or call **877 788 1202** toll free.

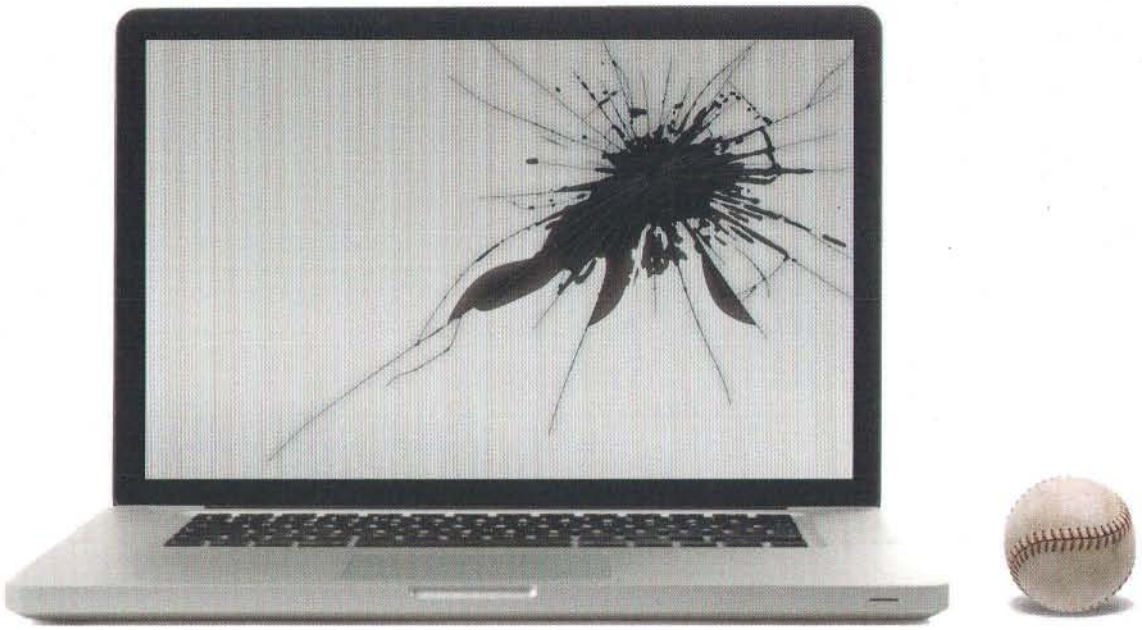
```
    — prepare the git-config statement
    set tCmd to "cd " & aPth & ";"
    set tCmd to tCmd & pGit & "-get " & aKey

    — execute the statement
    try
        set tVal to (do shell script tCmd)
    on error
        set tVal to "n/a"
    end try

    — return the key value
    return (tVal)
end readConfig — at aPth for aKey
```

Now the `loadGitProject()` routine shares the same code structure as `loadGitUser()`. The `loadGitColors()` routine (Listing 7), however, is a bit more complex due to the data it handles. First, the routine prepares two local records. The local `tGit` holds the color settings for three report types and the local `tSet` holds the color settings for each type. The routine then calls the `readConfig()` function, passing along the settings key for a given report. It then writes the returned color state into the `tGit` local. Next, the routine calls the function `readColorSlots()`. To that function, it passes the repository path, the report type and a record of slots. This function returns a list of color settings, which `loadGitProject()` then stores into the `tSet` local. Finally, it appends a copy of `tSet` to the `tGit` local. The routine repeats the same sequence of steps for the remaining types.

Listing 7. Reading a color setting.

```
to loadGitColors from aPth
    local tGit, tSet, tSlt, tVal

    — initialize the following locals
    set tGit to {diff:false, twig:false, stat:false}
    set tSet to {enbl:false, slot:{}, colr:{}}

    — read the following settings
    — color:git-branch
    set tSlt to {"current", "local", "remote", "plain"}

    set tVal to readConfig at aPth for "color.branch"
    if (tVal = "n/a") then
        set enbl of tSet to false
    else
        set enbl of tSet to tVal as boolean
    end if — (tVal = "n/a")

    set tVal to readColorSlots ¬
        from aPth for "branch" given slot:tSlt
    set slot of tSet to tSlt
    set colr of tSet to tVal
    copy tSet to twig of tGit

    — color:git-diff
    — ...truncated for length

    — color:git-status
    — ...truncated for length

    — return the retrieval results
    return (tGit)
end loadGitColors — from aPth
```

Listing 8 shows the code behind the `readColorSlots()`

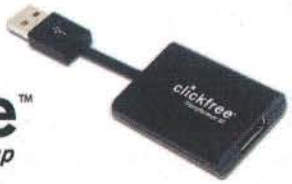


WHEN LIFE THROWS YOU A CURVE BALL.

Clickfree's Transformer SE Backup Adapter converts any brand USB hard drive, iPod or iPhone into an Award-Winning, Clickfree Automatic Backup device.

Watch the demo video -

clickfree™
Automatic Backup



1 Plug it into your Computer with any external hard drive

2 Backup Starts Automatically - Your Files Are Safe!

routine. This routine begins with its `tLst` local set to an empty list. It then parses each entry of its `aSlt` input and passes the result, a slot name, to the `readColorValue()` function. That function takes the slot name and adds it to the `git-config` statement. It submits the statement to `readConfig()` and gets a color value in return. If it gets an `'n/a'` result, `readColorValue()` returns a default color value of `'normal'`. Whatever color value is returned, `readColorSlots()` adds the value to the end of its `tLst` local. This ensures that each color value is in the same position as the slot.

Listing 8. Reading the colors of each report slot.

```
to readColorSlots from aPth for aCmd given slot:aSlt
    local tSlt, tLst, tGit, tHue

    — initialize the following locals
    set tLst to {}

    — start parsing the color slots
    repeat with tSlt in aSlt
        set tHue to (readColorValue from aPth ¬
            given command:aCmd, slot:tSlt)
        copy tHue to the end of tLst
    end repeat — with tSlt in aSlt

    — return the retrieval results
    return (tLst)
end readColorSlots — from aPth for aCmd given slot:aSlt

to readColorValue from aPth given command:aCmd, slot:aSlt
    local tHue, tGit

    — prepare the Git subcommand
    set tGit to "color." & aCmd
    set tGit to tGit & "." & aSlt

    — read the color setting
    set tHue to readConfig at aPth for tGit
    if (tHue = "n/a") then
        set tHue to "normal"
    end if — (tHue = "n/a")

    — return the color value
    return (tHue)
end readColorValue — from aPth given command:aCmd, slot:aSlt
```

Displaying the settings

In order for GitConfig to display its lists of settings, it needs to know which widget gets which setting. This means each panel widget on the GitConfig window must have a unique name. Consider the User panel, for example (Figure 4). Each text field widget gets a four-character name. To assign a name, open the `MainMenu.nib` bundle with Interface Builder. Select a widget and choose AppleScript Inspector from the Tools menu. Then in the Name field of the ensuing inspector palette, type the name for that widget. Repeat the same steps for all relevant widgets.

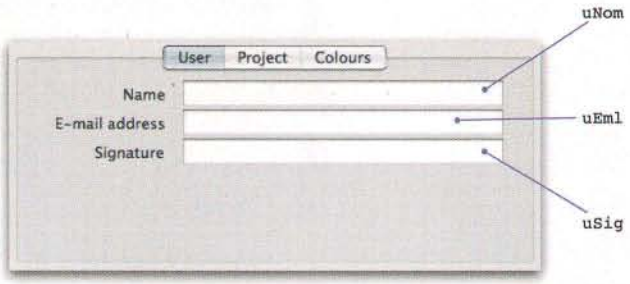


Figure 4. Naming the interface widgets.

Displaying each set of Git settings presents other challenges. First, not all widgets on each tab panel view are available after launch. If a panel is inactive after launch, the window *does not load* its widgets from the nib bundle. Thus, any references to that panel's widgets will point to a *null object*. Second, switching from one tab panel to another does cause the window to load the widgets that panel needs to display. On the other hand, that same window could *off-load* the widgets from the last active panel to save resources. This again results in that panel's widget references to point to null objects.

To detect when each widget becomes available, GitConfig relies on its `awake-from-nib` handler (Listing 9). Above the handler are six sets of global variables—they hold the references to each loaded widget. The handler starts by reading the class and name of the target widget. It then uses this information to store the widget to the correct global. Note the same handler also detects and stores a reference to the main window itself.

Listing 9. Detecting the widgets.

```
global gWin
global uNom, uEml, uSig
global rLnk, rEOL, rZip, zipL, rCln
global dChk, dSlt, dCol
global bChk, bSlt, bCol
global sChk, sSlt, sCol

on awake from nib anObj
    — identify the control widget
    set tTyp to class of anObj as string
    set tNom to name of anObj as string

    if (tTyp is "tab view item") then
        — view:panel:tab
        — RESERVED

    else if (tTyp is "text field") then
        — control:field:text
        try
            if (tNom is "uNom") then
                — control:field:text:user:name
                set uNom to anObj
            else if (tNom is "uEml") then
                — control:field:text:user:e-mail
                set uEml to anObj
            else if (tNom is "uSig") then
                — control:field:text:user:signature
                set uSig to anObj
            end if — (tNom is "uNom")
        on error eMsg
            log ("[GitConfig:awake-from-nib:" & eMsg)
        end try
```

casper SUITE

JAMF Software is committed to helping customers increase their understanding about the ways in which the Casper Suite can be used to address common IT challenges. In the coming months we'll be hosting a series of events focused around the ways in which the Casper Suite can be leveraged to achieve mature solutions for such common tasks as Malware and Anti-Virus protection, Backup and Data Integrity, User State Management, and Full Disk Encryption.

New Event Announcement

Securing the Mac OS: Data Protection and Backup using CrashPlan PRO

Thursday July 15, 2010 2pm Central Time

Register for this free event:
jamfsoftware.com/solutions/backup

Apple Mobile Devices in Enterprise Environments
Archived: jamfsoftware.com/solutions/iphone

One to One Apple Deployments with the Casper Suite
Archived: jamfsoftware.com/solutions/one-to-one

Configuring the Mac OS for Secure Environments
Archived: jamfsoftware.com/solutions/security

Securing the Mac OS: Data Protection and Backup using CrashPlan PRO
Thursday July 22, 2010 2pm
Registration: jamfsoftware.com/solutions/backup

Anti-Virus Protection
August 2010

User State Management
September 2010

Full Disk Encryption
October 2010

Complete product and event information at **jamfsoftware.com**

Configuring the Mac OS for secure environments

As Enterprise businesses increasingly adopt Mac OS computers, regulatory compliance has grown in importance on the platform. While Windows system administrators have long had a variety of solutions from which to choose, the Mac system admin faces a lack of mature tools that Alert, Report and Remediate Control Objectives. The Casper Suite fills this gap by combining technologies that are native to the platform (Mac OS) with a management framework (The Casper Suite) that automates many of the processes.

Whether you are a Government, Commercial or Education organization, you will benefit from this information about using native tools within the Mac OS and the Casper Suite to configure your server, enforce security standards and create custom reports for demonstrating regulatory compliance.

View the archived presentation and read more about Secure Environments and regulatory compliance online:

http://www.jamfsoftware.com/solutions/security

JAMF software

```
    else if (tTyp is "popup button") then
        — control:button:pop-up
        if (tNom is "zipL") then
            — control:button:pop-up:compression:level
            set zipL to anObj
        else if (tNom is "dSlt") then
            —
            — ... truncated for length
            —
        end if — (tNom is "uNom")

    else if (tTyp is "button") then
        — control:button:check-box
        if (tNom is "rLnk") then
            — control:button:check-box:symlink:follow
            set rLnk to anObj
        else if (tNom is "rEOL") then
            — control:button:check-box:end-of-line:POSIX
            set rEOL to anObj
        else if (tNom is "rZip") then
            — control:button:check-box:compression:enable
            set rZip to anObj
        else if (tNom is "rCln") then
            — control:button:check-box:clean:explicit
            set rCln to anObj
        else if (tNom is "dChk") then
            —
            — ... truncated for length
            —
        end if — (tNom is "uNom")

    else if (tTyp is "window") then
        — view:window
        if (tNom is "617C") then
            set gWin to anObj
        end if — (tNom is "617C")
    end if — (tTyp is "tab view item")
end awake from nib - anObj
```

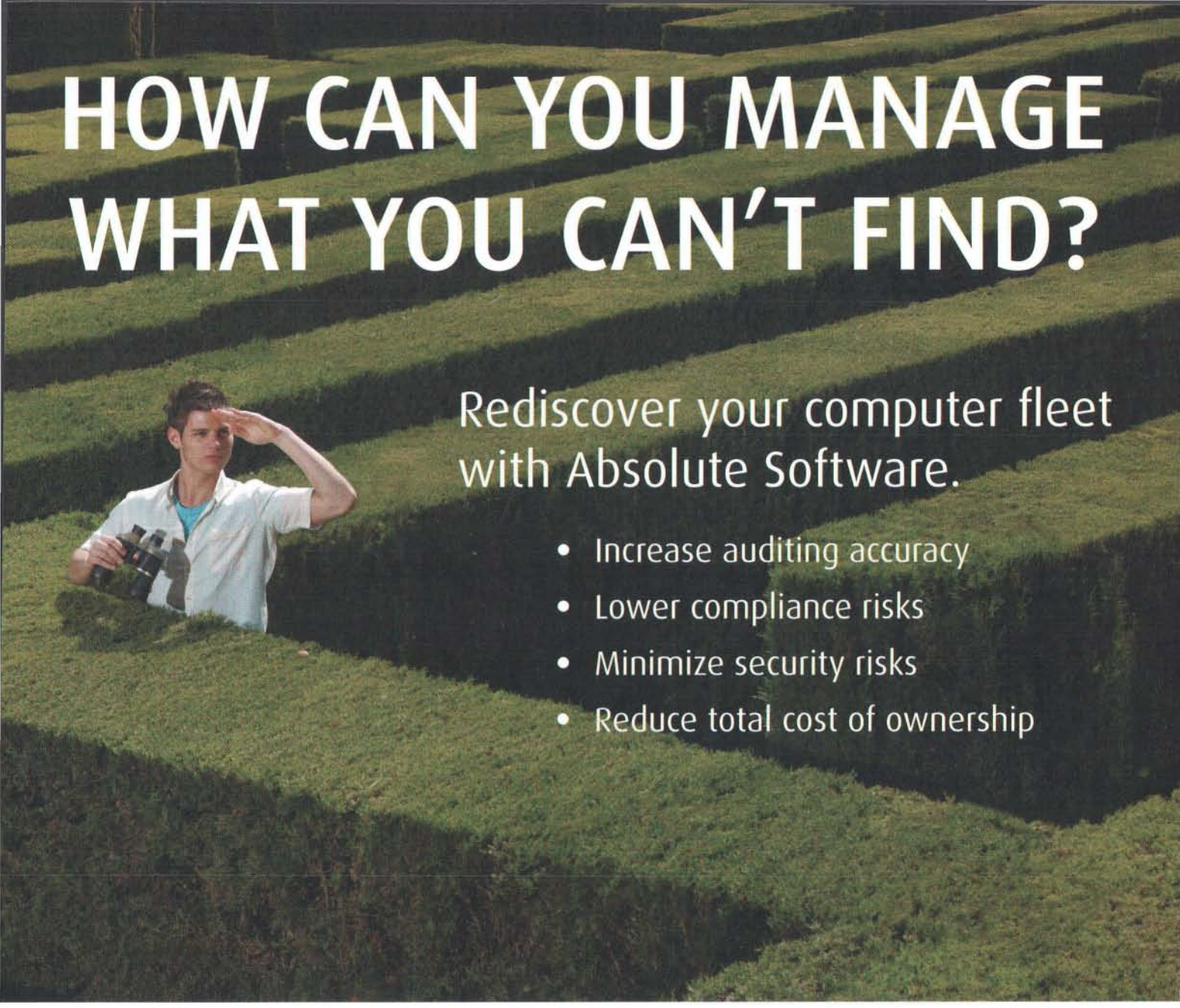
For the `awake-from-nib` handler to do its job, it must be attached to each widget on the GitConfig window. This too is easily done in Interface Builder. Select again each widget and choose AppleScript Inspector from the Tools menu. From the Scope pop-up menu, choose the menu item Global. From the Script pop-up, choose `GitConfig.applescript`. Then from the Event Handlers list, set the checkbox next to the `awake-from-nib` event. Save your changes when done.

Now the actual display is handled by three separate routines, one for each group of settings. Each routine gets one input argument, `aCfg`, which is the record object holding the settings. The `showGitUser()` routine (Listing 10) handles the user settings. This one simply writes the settings to the correct widget on the User panel. The `showGitProject()` routine handles the repository settings. It too writes the settings to the widgets on the Repository panel. But in the case of `core.compresion`, the routine sets and enables the `rZip` outlet if the compression level is greater than zero. Then it displays the level on the `zipL` outlet.

Listing 10. Displaying the user and repository settings.

```
to showGitUser from aCfg
    try
        set content of uNom to uNom of aCfg
        set content of uEml to uEml of aCfg
        set content of uSig to uSig of aCfg
    on error eMsg
        log ("[GitConfig] showGitUser:" & eMsg)
    end try
```

HOW CAN YOU MANAGE WHAT YOU CAN'T FIND?



Rediscover your computer fleet with Absolute Software.

- Increase auditing accuracy
- Lower compliance risks
- Minimize security risks
- Reduce total cost of ownership

Find and manage your Mac and PC computers with Absolute® Software. From the largest corporations to your home office, our Computrace®, Absolute Manage and LoJack® for Laptops solutions help you improve data protection, simplify computer lifecycle management and recover stolen computers.

For a FREE demo of Absolute Manage visit:

www.absolute.com/rediscover

Absolute®Software

The absolute best way to track, manage & protect your digital world.

```
end showGitUser — from aCfg

to showGitProject from aCfg
    local tVal, tFlg

    try
        set state of rLnk to (rLnk of aCfg as boolean)
        set state of rEOL to (rEOL of aCfg as boolean)
        set state of rCln to (rCln of aCfg as boolean)

        set tFlg to (rZip of aCfg as boolean)
        if (tFlg) then
            set tVal to zipL of aCfg as string
            if (tVal is "9") then
                set tVal to tVal & " (highest)"
            end if — (tVal is "9")
        else
            set tVal to "0 (none)"
        end if — (tFlg)

        set state of rZip to tFlg
        set enabled of zipL to tFlg
        set title of zipL to tVal

    on error eMsg
        log ("[GitConfig] showGitProject:" & eMsg)
    end try
end showGitProject — from aCfg
```

Not surprisingly, the `showGitColors()` (Listing 11) routine takes extra steps to ensure each report slot is mapped to the right color. It starts by reading the report's color settings from the `aCfg` argument. Then it displays that report's color state stored under the `enbl` record key. Next, it reads the displayed slot name from the outlet linked to the pop-up menu widget (shown here as `dSlt`). The routine then calls the `getSlotColor()` function, passing as input the slot name and the color settings. This function returns the color value for the given slot. Then `showGitColors()` displays the returned color on the second outlet (here being `dCol`). The routine follows the same series of steps for the other two Git reports.

Listing 11. Displaying the color settings.

```
to showGitColors from aCfg
    local tSet, tFlg, tSlt, tHue

    — display the following color settings
    — panel:settings:colors:diff
    set tSet to diff of aCfg
    set tFlg to (enbl of tSet as boolean)
    set state of dChk to tFlg
    set enabled of dSlt to tFlg
    set enabled of dCol to tFlg

    set tSlt to title of dSlt
    set tHue to getSlotColor for tSlt from tSet
    set title of dCol to (tHue as string)

    — panel:settings:colors:branch
    — ...truncated for length

    — panel:settings:colors:status
    — ...truncated for length
end showGitColors — from aCfg

to getSlotColor for aSlt from aSet
    local tHue, tLst, tSlt
    local tIdx, tLen

    — set the default color setting
    set tHue to "normal"

    try
        — locate the slot index
        set tLst to slot of aSet
        set tLen to length of tLst
        repeat with tIdx from 1 to tLen
            set tSlt to item tIdx of tLst
            if (tSlt is aSlt) then
                exit repeat
            end if — (tSlt is aSlt)
        end repeat — with tIdx from 1 to tLen

        — read the color value
        set tLst to colr of aSet
        set tHue to item tIdx of tLst
    on error eMsg
        — RESERVED
    end try

    — return the color setting
    return (tHue)
end getSlotColor — for aSlt from aSet
```

The GitConfig tool invokes each display routine at different points of its process cycle. For instance, it invokes only the `showGitUser()` routine after it displays its main window (Listing 12). This is because the window has the **User** tab panel as its default panel view. When users switches to a different panel view, GitConfig uses the `selected-tab-view-item` handler to invoke the display routine for that panel. Notice how both handlers keep track of the active panel view with the `gPnl` global.

Listing 12. Invoking a display routine.

```
global gPnl

on launched anApp
    — display the user settings
    set gPnl to "tUsr"
    showGitUser from gUsr
end launched — anApp

on selected tab view item aTab tab view item aPnl
    — identify the active panel
    set gPnl to name of aPnl
    if (gPnl is "tUsr") then
        — panel:settings:user
        showGitUser from gUsr
    else if (gPnl is "tPrj") then
        — panel:settings:project
        showGitProject from gPrj
    else if (gPnl is "tCol") then
        — panel:settings:colors
        showGitColors from gHue
    end if — (gPnl is "tUsr")
end selected tab view item — aTab tab view item aPnl
```

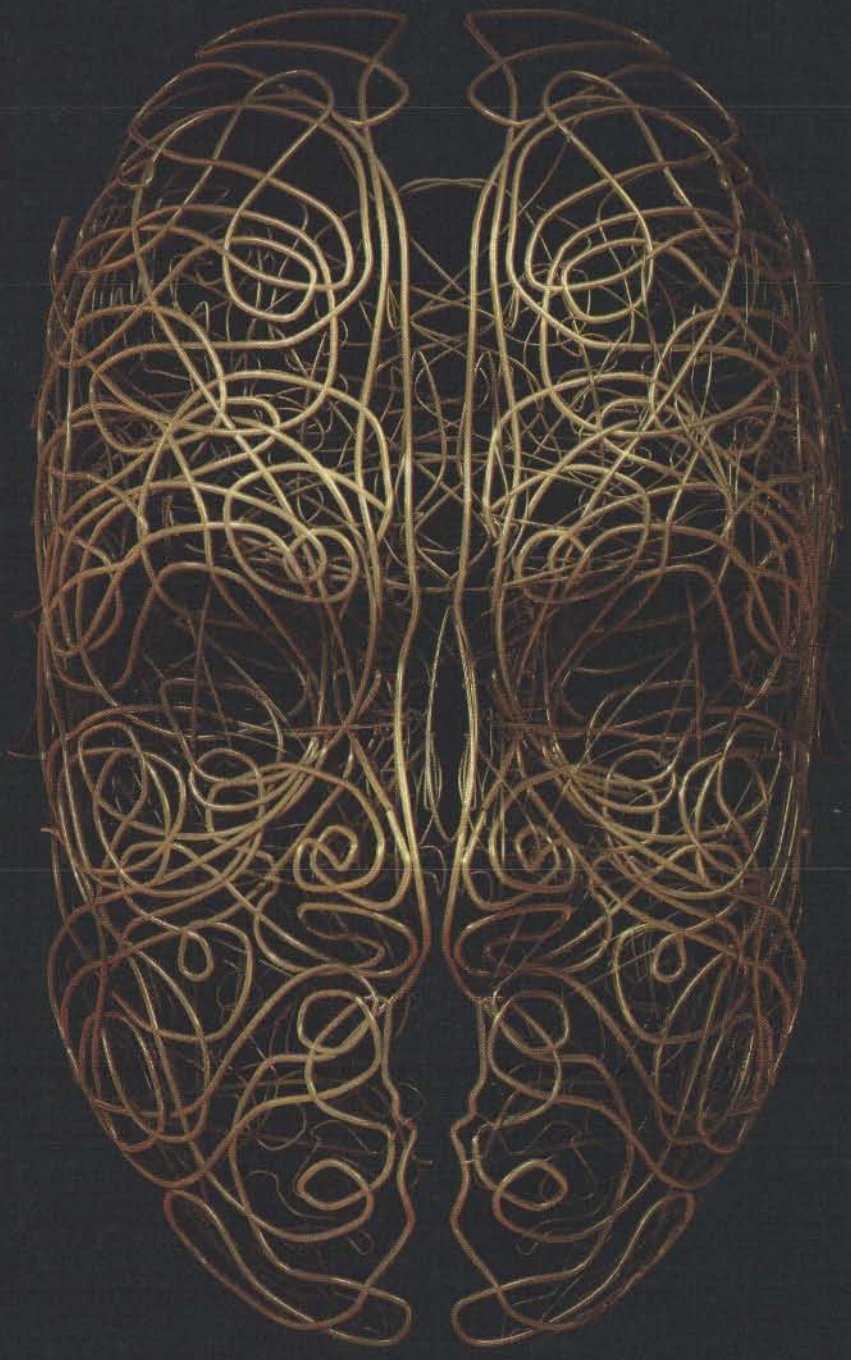
As for the **Colors** panel, GitConfig uses the `choose-menu-item` handler (Listing 13) to detect which report slot is selected by the user. Here it checks if the calling object is a pop-up button widget. Then it checks that widget's unique name. Next, it reads the chosen slot name. It passes that name, together with the correct color record, to the `getSlotColor()` function to retrieve the color value. And then it passes the color value to the second pop-up button widget for display.

More than just a pretty face.

Flesh out the concept with 3D characters

Let the technical go and your creativity flow

Conquer the challenge in record time



POSER® PRO 2010

Professional 3D Figure Design & Animation

Poser Pro 2010 is the most efficient way to include 3D figures in your next creative project. With over 2.5GB of ready to use assets, pose and animate beautifully sculpted 3D characters, without needing to model and rig them. Plug-in to 3ds Max, Maya, CINEMA 4D and LightWave, or use COLLADA to bring Poser content into your favorite production tools. It's easy. It's fast. It's creative. It's Poser Pro 2010. **Visit poser.smithmicro.com**

smithmicro software

Listing 13. Displaying a color value.

```
on choose menu item aMnu
    local tNom, tTyp, tSet, tPth
    local tVal, tChg

    — initialize the following locals
    set tNom to name of aMnu as string
    set tTyp to class of aMnu as string

    — identify the calling widget
    if (tTyp is "menu item") then
        — menu:file
        — ...truncated for length

    else if (tTyp is "popup button") then
        — menu:pop-up
        set tChg to true

        - identify the popup button widget
        if (tNom is "zipL") then
            — menu:pop-up:compression:level
            — ...truncated for length

        else if (tNom is "dSlt") then
            — menu:pop-up:diff:slot
            — load the slot color setting
            set tNom to title of dSlt
            set tSet to diff of gHue

            set tNom to getSlotColor for tNom from tSet
            set title of dCol to (tNom as string)

        else if (tNom is "dCol") then
            — menu:pop-up:diff:color
            — ...truncated for length

        else if (tNom is "bSlt") then
            — menu:pop-up:branch:slot
            — load the slot color setting
            set tNom to title of bSlt
            set tSet to twig of gHue

            set tNom to getSlotColor for tNom from tSet
            set title of bCol to (tNom as string)

        else if (tNom is "bCol") then
            — menu:pop-up:branch:color
            — ...truncated for length

        else if (tNom is "sSlt") then
            — menu:pop-up:status:slot
            — load the slot color setting
            set tNom to title of sSlt
            set tSet to stat of gHue

            set tNom to getSlotColor for tNom from tSet
            set title of sCol to (tNom as string)

        else if (tNom is "sCol") then
            — menu:pop-up:status:color
            — ...truncated for length
        end if — (tNom is "zipL")

        — set the window changed state
        set document edited of gWin to tChg
    end if — (tTyp is "menu item")
end choose menu item — aMnu
```

Updating the settings

Now when users change one or more settings on each GitConfig panel, GitConfig writes the changes back to its *global buffers*, but not to the settings file. This preserves the original settings still in the files and allows users to reverse their changes.

How GitConfig handles the changes depends on the widget that held the change. If the widget is a text field widget, GitConfig uses the `changed` handler (Listing 14). This handler reads the content and unique name of the widget. Then it writes the content to the correct record item in the `gUsr` global.

Listing 14. Handling the text field widget.

```
on changed anObj
    local tVal, tNom

    — initialize the following locals
    set tNom to name of anObj
    set tVal to content of anObj

    — identify the calling widget
    if (tNom is "uNom") then
        set uNom of gUsr to tVal
    else if (tNom is "uEml") then
        set uEml of gUsr to tVal
    else if (tNom is "uSig") then
        set uSig of gUsr to tVal
    end if — (tNom is "")

    — set the window changed state
    set document edited of gWin to true
end changed — anObj
```

If the changed widget is a checkbox button, GitConfig uses the `clicked` handler to deal with the change (Listing 15). This handler also begins by reading the name and state of the widget. Based on the name, it stores the retrieved state into the record entry of the correct global. But for the widget named `rZip`, the handler displays the latest compression level for the repository.

Listing 15. Handling the checkbox button.

```
on clicked anObj
    local tNom, tFlg

    — initialize the following locals
    set tNom to name of anObj
    set tFlg to state of anObj as boolean

    — identify the calling widget
    if (tNom is "rLnk") then
        — setting:flag:project:link
        set rLnk of gPrj to tFlg

    else if (tNom is "rEOL") then
        — setting:flag:project:end-of-line
        set rEOL of gPrj to tFlg

    else if (tNom is "rZip") then
        — setting:flag:project:compression
        set enabled of zipL to tFlg
        if (tFlg) then
            set tNom to zipL of gPrj as string
            if (tNom is "9") then
                set tNom to tNom & " (highest)"
            else if (tNom is "0") then
                set tNom to tNom & " (none)"
            end if — (tNom is "9")

            set title of zipL to tNom
        else
            set title of zipL to "0 (none)"
            set zipL of gPrj to 0
        end if — (tFlg)
        set rZip of gPrj to tFlg

    else if (tNom is "rCln") then
        — setting:flag:project:clean:explicit
        set rCln of gPrj to tFlg
```

Made for iPod iPhone

IOGEAR

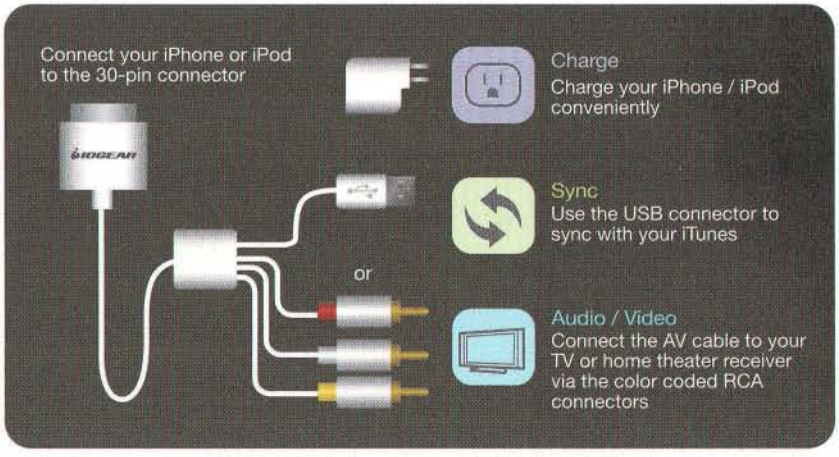
Watch Your iPod or iPhone on Your TV

Enjoy audio and video content from your iPhone or iPod on the big screen, no more crowding around a small screen

Composite AV Cable
with Charge & Sync for iPhone / iPod

Model: GIPODAVC6

- Connect an iPod or iPhone to your TV, gather up your friends and watch videos or slideshows together on the big screen
- Enjoy a room-filling audio experience with your stereo or powered speakers
- Plug the USB connector into a power source to charge while viewing
- Conveniently sync your iPod or iPhone with your computer
- Ultra-shielded copper cables provides premium audio and video quality



iPhone 3GS
16GB 32GB

iPhone 3G
8GB 16GB

iPhone
4GB 8GB 16GB

iPod touch
2nd generation
8GB 16GB 32GB 64GB

iPod touch
1st generation
8GB 16GB 32GB

iPod
5th generation (video)
30GB

iPod classic
120GB 160GB (2009)

iPod nano
4th generation (video)
8GB 16GB

iPod nano
1st generation
1GB 2GB 4GB

"Made for iPod" and "Made for iPhone" mean that an electronic accessory has been designed to connect specifically to iPod and iPhone, respectively, and has been certified by the developer to meet Apple performance standards. Apple is not responsible for the operation of this device or its compliance with safety and regulatory standards.

www.iogear.com

```
    else if (tNom is "dChk") then
        — setting:flag:color:diff
        set enabled of dSlt to tFlg
        set enabled of dCol to tFlg
        set enbl of diff of gHue to tFlg

    else if (tNom is "bChk") then
        — setting:flag:color:branch
        set enabled of bSlt to tFlg
        set enabled of bCol to tFlg
        set enbl of twig of gHue to tFlg

    else if (tNom is "sChk") then
        — setting:flag:color:status
        set enabled of sSlt to tFlg
        set enabled of sCol to tFlg
        set enbl of stat of gHue to tFlg

    end if — (tNom is "rLnk")

    — set the window changed state
    set document edited of gWin to true
end clicked — anObj
```

Supposed the changed widget is a pop-up button? In this case, GitConfig handles the change via its `choose-menu-item` handler (Listing 16). This is the same handler that processes the File menu selections. It is also the same handler that displays the color values for each report slot.

Listing 16. Handling a pop-up menu.

```
on choose menu item aMnu
    local tNom, tTyp, tSet, tPth
    local tVal, tChg

    — initialize the following locals
    set tNom to name of aMnu as string
    set tTyp to class of aMnu as string

    — identify the calling widget
    if (tTyp is "menu item") then
        — menu:file
        — ...truncated for length

    else if (tTyp is "popup button") then
        — menu:pop-up
        set tChg to true

        — identify the popup button widget
        if (tNom is "zipL") then
            — menu:pop-up:compression:level
            set tVal to title of aMnu
            set tVal to character 1 of tVal
            set zipL of gPrj to tVal as integer
            log gPrj

        else if (tNom is "dSlt") then
            — menu:pop-up:diff:slot
            — ...truncated for length

        else if (tNom is "dCol") then
            — menu:pop-up:diff:color
            — read the color setting
            set tNom to title of dCol
            set tTyp to title of dSlt

            — update the settings buffer
            set tSet to diff of gHue
            set tSet to (setSlotColor on tSet for tTyp into
tNom)
            set diff of gHue to tSet

        else if (tNom is "bSlt") then
            — menu:pop-up:branch:slot
```

audioengine

upgrade your music!

GET 10% OFF plus free shipping when you buy online! Use Coupon Code **MTTEN**

Experience our award-winning sound, high-quality materials and truly useful features.

$199 per pair

Audioengine 2 (A2)
Premium Powered Desktop Speakers

Closes the gap between computer speakers and home audio

*"These **are** the best speakers for your desktop, computer, or media player."* – Connect Reviews

$119 per set

Audioengine W2 (AW2)
Premium Wireless Adapter for iPod

Unwire your iPod

"Super fast setup and the uncompressed sound is pretty remarkable." – Uncrate

Model A5N shown in Solid Carbonized Bamboo $449

starting at $349 per pair

Audioengine 5 (A5)
Premium Powered Bookshelf Speakers

High-quality audio for your Mac or PC

"There are no other speakers in this price range that come close." – Mac Observer

$99 per set

Audioengine W1 (AW1)
Premium Wireless Audio Adapter

Send CD-quality wireless anywhere with or without a computer

"High quality wireless music streaming that's quick and easy to use." – Register Hardware

Works with all your gear · All cables included · 30-day audition · 3 year warranty

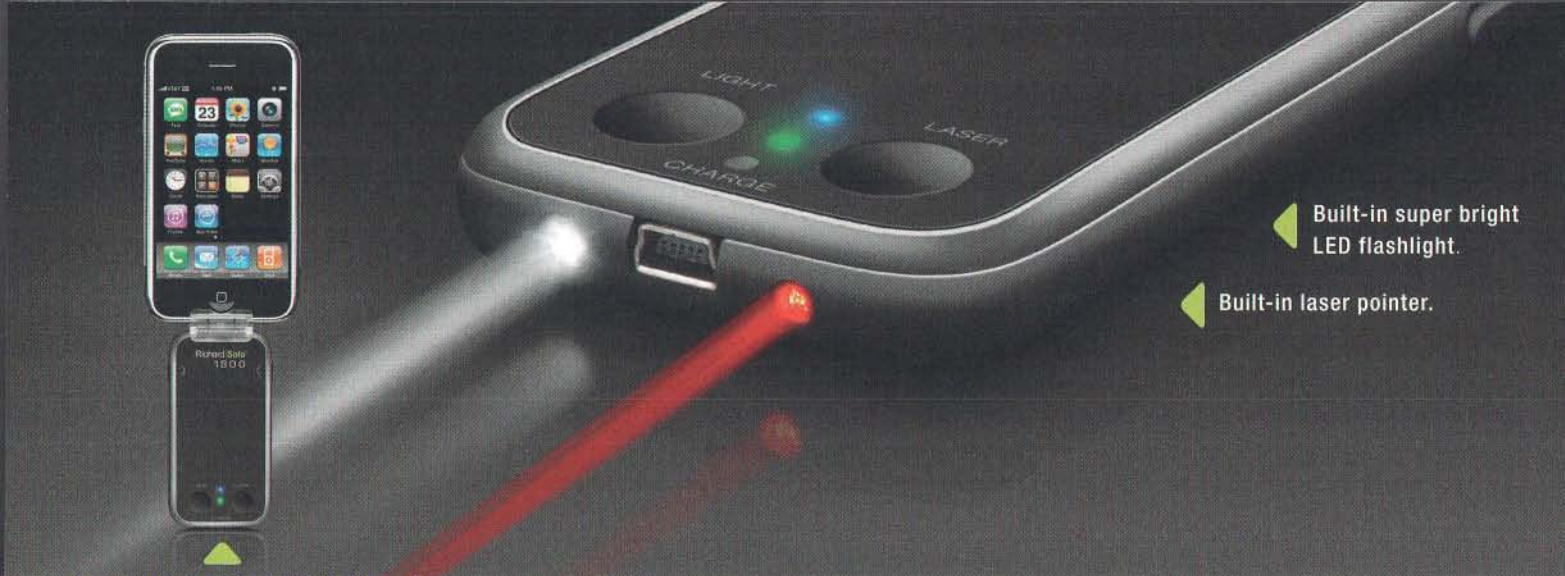
Visit our website for more product info, reviews, and awards: **www.audioengineusa.com**

```
            — ...truncated for length

        else if (tNom is "bCol") then
            — menu:pop-up:branch:color
            — read the color setting
            set tNom to title of bCol
            set tTyp to title of bSlt

            — update the settings buffer
            set tSet to twig of gHue
            set tSet to (setSlotColor on tSet for tTyp into
tNom)
            set twig of gHue to tSet

        else if (tNom is "sSlt") then
            — menu:pop-up:status:slot
            — ...truncated for length

        else if (tNom is "sCol") then
            — menu:pop-up:status:color
            — read the color setting
            set tNom to title of sCol
            set tTyp to title of sSlt

            — update the settings buffer
            set tSet to stat of gHue
            set tSet to (setSlotColor on tSet for tTyp into
tNom)
            set stat of gHue to tSet
        end if — (tNom is "zipL")

        — set the window changed state
        set document edited of gWin to tChg
    end if — (tTyp is "menu item")
end choose menu item — aMnu
```

As before, the handler reads the name and type of the affected widget. If the widget is a menu item, the handler runs the code as described in Listings 4, 19 and 22. But if the widget is a pop-up button and one that displays a list of report slots, the handler runs the code in Listing 12. Now if the widget is one that displays a list of color values, the handler reads the report slot that corresponds to that color value. Next, it reads the chosen color value. Finally, the handler calls the `setSlotColor()` routine (Listing 17), passing for input the slot and color, as well as the record entry for the given report. The routine then returns the modified record, which the handler stores into the `gHue` global.

Listing 17. Setting the color value.

```
to setSlotColor on aSet for aSlt into aHue
    local tHue, tSlt, tLst, tMod
    local tIdx, tLen

    — initialize the following local
    copy aSet to tMod

    try
        — locate the slot index
        set tLst to slot of tMod
        set tLen to length of tLst
        repeat with tIdx from 1 to tLen
            set tSlt to item tIdx of tLst
            if (tSlt is aSlt) then
                — update the slot color
                set tLst to colr of tMod
                set item tIdx of tLst to aHue
                set colr of tMod to tLst
            end if —(tSlt is aSlt)
        end repeat — with tIdx from 1 to tLen
    on error
        — RESERVED
```



For iPhone 2G/3G/3GS/iPod
Direct plug-in / no cable.
Includes 2G / 3G support brace.
RS001 ~~$69.95~~ **$39.95**

For iPhone/iPod with Cable
Cable connection only.
Perfect for iPod touch.
RS008 ~~$69.95~~ **$39.95**

For All BlackBerry/Smartphones
Cable or direct plug-in.
Works with all USB-port phones.
RS007 ~~$69.95~~ **$39.95**

10 Reasons RichardSolo 1800 is the best backup battery for iPhone/ Smartphones — and *MacTech* readers save $30. You pay only $39.95

1. It is the only one that actually "latches" onto the iPhone — very stable.
2. Includes free, slim, protective hard case for iPhone 3G/3GS ($24.95 value) that works perfectly with included support brace.
3. Unlike "slipcase" configurations, there is no rear blockage of your cell phone antenna.
4. Licensed and certified by Apple for iPhone 2G/3G/3GS/iPod.

   Made for iPod Works with iPhone
5. Built-in flashlight is surprisingly useful and bright; laser pointer included.
6. Lightweight — you can easily carry it in your pocket, and top up your iPhone and iPod as needed.
7. Choose from two models: direct plug-in or cable. Both are 1800 mAh lithium-ion rechargeable!
8. We support you. Quick email response by the best customer service in the industry, and a full 1-year warranty.
9. **Free bonus** car charger and wall charger included. Charge battery and iPhone in tandem!
10. *MacTech* special price; enter the discount code ***MacTech*** at checkout. Instantly save $30!

Online ordering and blog reviews: www.RichardSolo.com

Enter the discount code *MacTech* at checkout to instantly save $30. You pay only $39.95

Offer cannot be combined with any other discounts or promotions.

Weekly GREAT DEAL Sign up today at RichardSolo.com to receive our weekly great deal email offer!

Order now online: www.RichardSolo.com

Richard|Solo®

```
    end try

    — return the modified settings
    return (tMod)
end setSlotColor — on aSet for aSlt into aHue
```

Committing the settings

Now when users choose Save Settings from the File menu, GitConfig reacts by invoking its `choose-menu-item` handler (Listing 18). This handler runs three save routines when the menu item's signature is `'save'`.

Listing 18. Handling the Save Settings menu item.

```
on choose menu item aMnu
    local tNom, tTyp, tSet, tPth
    local tVal, tChg

    — initialize the following locals
    set tNom to name of aMnu as string
    set tTyp to class of aMnu as string

    — identify the calling widget
    if (tTyp is "menu item") then
        — menu:file
        if (tNom is "slct") then
            — file:repository:settings:select
            — ...truncated for length

        else if (tNom is "save") then
            — file:repository:settings:save
            — save the following settings
            — repository:settings:user
            saveGitUser from gUsr

            — repository:settings:project
            saveGitProject from gPrj

            — repository:settings:colors
            saveGitColors from gHue

            — clear the window changed state
            set document edited of gWin to false
        else if (tNom is "rvrt") then
            — file:repository:settings:revert
            — ...truncated for length

    else if (tTyp is "popup button") then
        — menu:pop-up
        — ...truncated for length

    end if — (tTyp is "menu item")
end choose menu item — aMnu
```

The first save routine `saveGitUser()` (Listing 19) writes the settings held in the `gUsr` global. It reads each entry from `gUsr` and passes them to the `writeConfig()` routine. It also passes the settings key and the location of the settings file. The `writeConfig()` routine uses the supplied key, value and path to prepare the `git-config` statement. It then executes the statement with the OSAX function `do-shell-script`.

Listing 19. Committing the user settings.

```
to saveGitUser from aCfg
    local tVal

    — save the following settings
    — settings:user:name
    set tVal to uNom of aCfg
    set tVal to "'" & tVal & "'"
```

- ☑ Android
- ☑ BlackBerry
- ☑ Palm Pre
- ☑ Symbian

Sync ☑ Windows Mobile with Mac

Wirelessly

Your smartphone frees you from the land-line phone in your office. Your Bluetooth headset frees you from cabled earphones and mic. Let The Missing Sync free you from tethering your phone to your computer every time you want to sync. Sync over Wi-Fi or Bluetooth.*

Automatically

When your smartphone is near your computer, sync happens without having to think about it. Photos you snap on your phone, changes to contacts, new appointments, new music - they'll sync automatically between your phone and computer. It's just like magic!**

www.markspace.com/SyncIt

mark/space

*Wireless capabilities vary among phones, so The Missing Sync features will vary, as well.

**Proximity Sync feature not yet available for The Missing Sync for Windows Mobile.

```
    writeConfig for "user.name" at gPth into tVal

    — settings:user:e-mail
    set tVal to uEml of aCfg
    set tVal to "'" & tVal & "'"
    writeConfig for "user.email" at gPth into tVal

    — settings:user:signature
    set tVal to uSig of aCfg
    set tVal to "'" & tVal & "'"
    writeConfig for "user.signingkey" at gPth into tVal
end saveGitUser — from aCfg

to writeConfig for aKey at aPth into aVal
    local tCmd, tErr

    — prepare the git-config statement
    set tCmd to "cd " & aPth & ";"
    set tCmd to tCmd & pGit & aKey & " " & aVal

    — execute the statement
    try
        set tErr to (do shell script tCmd)
    on error eMsg
        set tErr to eMsg
    end try
end writeConfig — for aKey at aPth into aVal
```

The second save routine `saveGitProject()` has the same code structure as `saveGitUser()`. But the `saveGitColors()` routine (Listing 20) uses a more complex structure to ensure each report slot is mapped to the right color. This routine begins by preparing the settings key for a given report. It then checks the color state for that report. If colors are enabled, the routine reads the slots and color entries, and then passes them to the `writeConfig()` routine. Afterwards, it passes the color state to the same `writeConfig()` routine.

Listing 20. Committing the report colors.

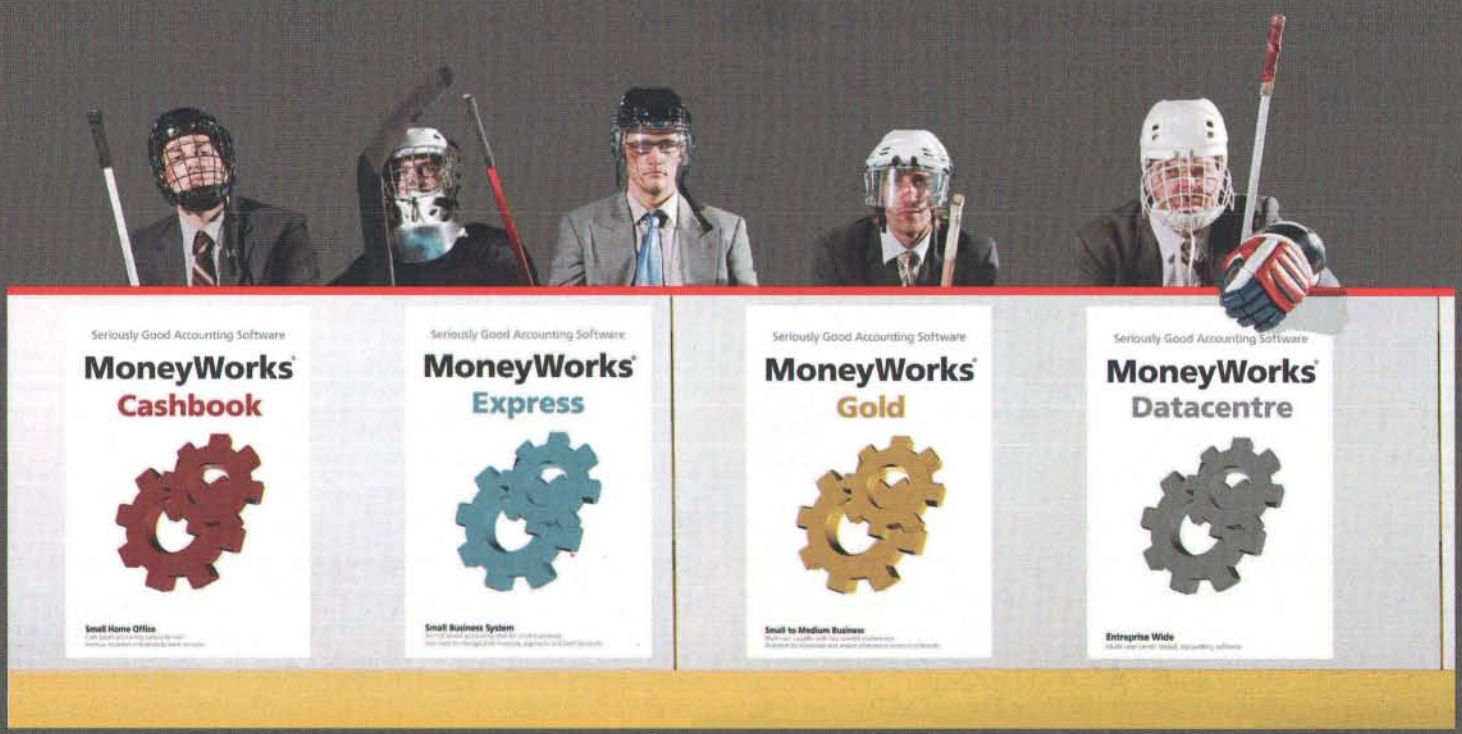
```
to saveGitColors from aCfg
    local tFlg, tSlt, tHue, tGit
    local tSet, tLst
    local tIdx, tMax

    — save the following settings
    — settings:project:color:branch
    set tGit to "color.branch"
    set tSet to twig of aCfg
    set tFlg to enbl of tSet
    if (tFlg) then
        — settings:project:color:branch:slots
        set tLst to slot of tSet
        set tMax to count of tLst

        repeat with tIdx from 1 to tMax
            set tSlt to item tIdx of slot of tSet
            set tSlt to tGit & "." & tSlt
            set tHue to item tIdx of colr of tSet

            writeConfig for tSlt at gPth into tHue
        end repeat — with tIdx from 1 to tMax
    end if — (tFlg)
    set tFlg to tFlg as string
    writeConfig for tGit at gPth into tFlg

    — settings:project:color:diff
    — ...truncated for length

    — settings:project:color:status
    — ...truncated for length
end saveGitColors — from aCfg
```

MoneyWorks®

Seriously Good Accounting Software

Our customers tell us that our Canadian support team is helpful and polite.

They are, if you keep them off the ice.



Give them a call at 800-818-6955. It's free.

Or if it is after office hours when they are out playing hockey, send an email to support@moneyworks.ca. Emails are also free.

Download a trial version at moneyworks.ca/us/downloads

Suppose the user choose Revert to Save from the File menu? In that case, GitConfig runs another part of its **choose-menu-item** handler (Listing 21). This part runs only when the menu item has a signature of **'rvrt'**. It calls the same three load routines and stores their results into the appropriate global variable. Next, it checks the active panel view on the GitConfig window. It then calls the correct display routine for that panel.

Listing 21. Reversing the changes.

```
on choose menu item aMnu
    local tNom, tTyp, tSet, tPth
    local tVal, tChg

    — initialize the following locals
    set tNom to name of aMnu as string
    set tTyp to class of aMnu as string

    — identify the calling widget
    if (tTyp is "menu item") then
        — menu:file
        if (tNom is "slct") then
            — file:repository:settings:select
            — ...truncated for length

        else if (tNom is "save") then
            — file:repository:settings:save
            — ...truncated for length

        else if (tNom is "rvrt") then
            — file:repository:settings:revert

            — read the following settings
            — repository:settings:user
            set gUsr to loadGitUser from gPth
            — repository:settings:project
            set gPrj to loadGitProject from gPth

            — repository:settings:colors
            set gHue to loadGitColors from gPth

            — identify the active panel
            if (gPnl is "tUsr") then
                — panel:settings:user
                showGitUser from gUsr
            else if (gPnl is "tPrj") then
                — panel:settings:project
                showGitProject from gPrj
            else if (gPnl is "tCol") then
                — panel:settings:colors
                showGitColors from gHue
            end if — (gPnl is "tUsr")

            — clear the window changed state
            set document edited of gWin to false
        end if — (tNom is "slct")

    else if (tTyp is "popup button") then
        — menu:pop-up
        — ...truncated for length

    end if — (tTyp is "menu item")
end choose menu item — aMnu
```

Summing Up

The **git-config** command gives us the means to prepare and configure our Git setup. In this article, we managed to wrap the command in a simple graphical tool. Our tool is built with the aid of Xcode and AppleScript Studio. It lets us select which project repository to configure. It can load the most often used settings and save any changes made to those settings. Plus, it can reverse those same changes.

And that ends today's study AppleScript, Xcode and the open-source Git tool. But stay tuned to this spot as we learn more about Git and its capabilities. Until then, I bid you good day.

Bibliography and References

Scott Chacon (editor). *The Git Community Book*, pp. 21–22. GitHub [Online]. Available: http://book.git-scm.com.

Johannes Schindelin. "git-config". *The Git Manual*, The Linux Kernel Archives [Online]. Available: http://www.kernel.org/pub/software/scm/git/docs/git-config.html.

Bor Kraljic. *The Git Guide*, SourceMage Wiki [Online]. Available: http://www.sourcemage.org/Git_Guide

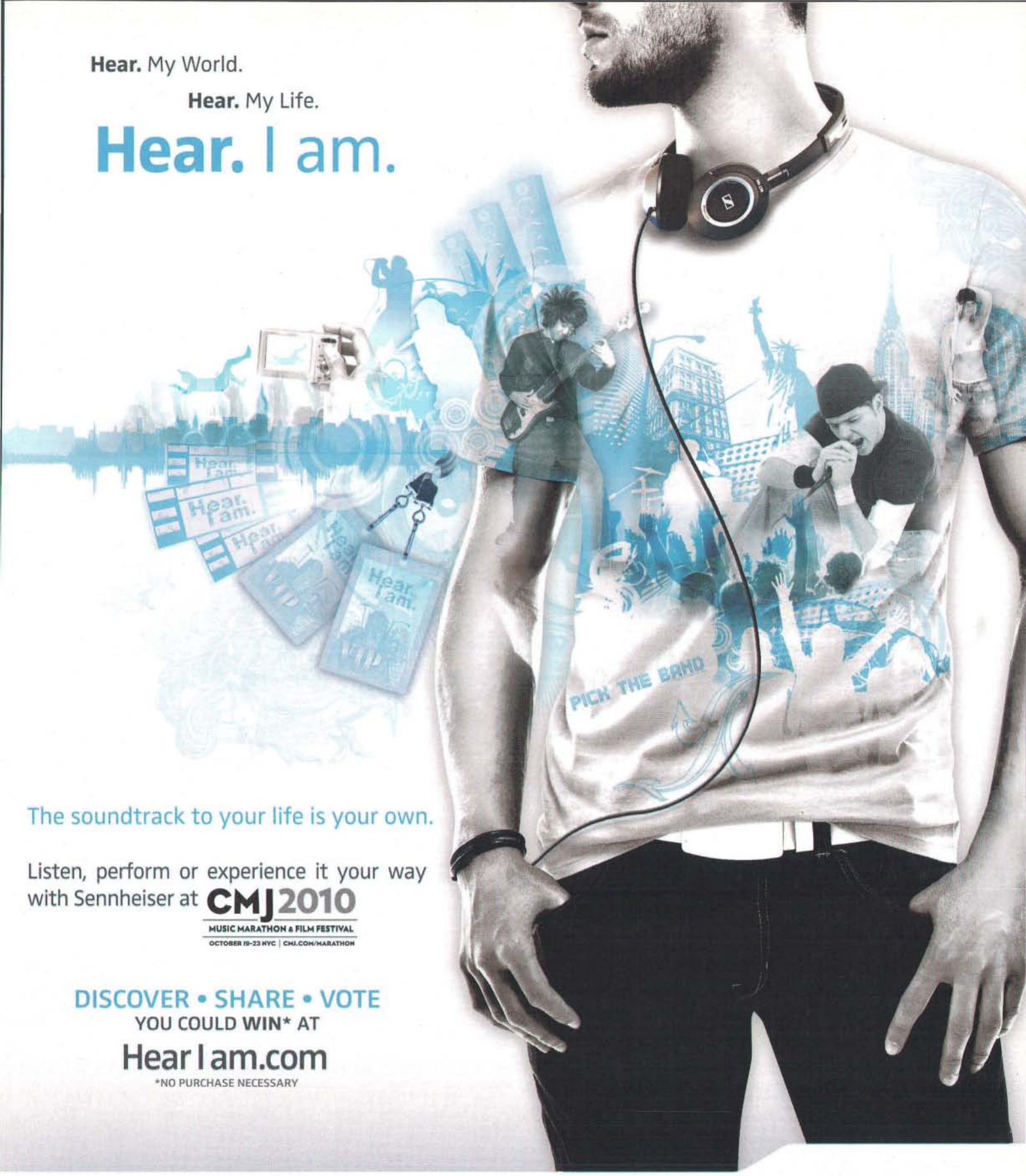
About The Author

JC is a freelance engineering writer from North Vancouver, British Columbia. He writes for various publications, covering diverse topics such as AppleScript, REALbasic, Python, and Cocoa. He also spends quality time with his foster nephew. You can reach JC at anarakisware@gmail.com.

Hear. My World.

Hear. My Life.

Hear. I am.



The soundtrack to your life is your own.

Listen, perform or experience it your way with Sennheiser at **CMJ 2010**

MUSIC MARATHON & FILM FESTIVAL

OCTOBER 19-23 NYC | CMJ.COM/MARATHON

DISCOVER • SHARE • VOTE

YOU COULD **WIN*** AT

HearIam.com

*NO PURCHASE NECESSARY

Headphones
& Microphones

SENNHEISER

by Edward Marczak

bash String Operations

Or, doing more with less

Welcome

At the end of last month's column I promised more bash tips and ticks this month. That's right: ticks. If I can have the latitude to define 'ticks' as, "a little used shall convention due to the nervous tick it induces," I think I can make that typo work retroactively. The bash shell has support for some of the same string operations that other languages do, without the need to call some non-native shell routine. This month, I'll introduce bash's string operators and examples of how to use them.

Variable Review

I've covered this in the past, but it's worth a review. Variable names consist of letters, digits and underscore characters. On assignment, just the variable name is needed:

```
greeting="Hello, "
```

When referencing a variable, it is preceded with the dollar-sign character to denote it as a variable to be tokenized for substitution:

```
echo "$greeting, $name"
```

However, rather than just a dollar-sign and the name, there's a more specific way to specify variables: enclosing them in curly-brackets. Get into the habit of enclosing *all* variables in curly brackets:

```
echo "${greeting}, ${name}"
```

This will save you headache, believe me. Why? First, you may need to assemble a file name like this:

```
filename=$user_$date.txt
```

In this case, the shell will try to concatenate "$user_" and "$date". Note the trailing underscore on "$user_" and you'll see why this will always be empty. This should be rewritten as:

```
filename=${user}_${date}.txt
```

Secondly, you'll need to use curly bracket variable syntax to reference any positional parameter variable greater than 9:
Right: `echo ${12}`
Wrong: `echo $11`
Without the curly brackets, the shell takes the shorter route and just uses $1. So, if the first positional parameter to a script were entered as "ralph", $10, $11 and $12 would be:

```
ralph0, ralph1, ralph2
```

Use the curly-bracket form to get the correct values.

Lastly, you'll need to use the curly brackets for anything used in the remainder of this article. Now, onto the article.

Operator, Operator!

Many development languages have straightforward ways of manipulating strings. Most people that use bash end up calling external programs to do this work. While that's sometimes appropriate, or even necessary, one can handle most chores with bash itself. bash has several built-in string operators, and I rarely see them used.

Each of the operations uses the curly-brace syntax. They work by adding special characters that are interpreted as operators.

Substitution Operators

The first group of operators allows for substitution of the supplied variable.

`${variable:-sub}` — Substitute if variable is null.
`${variable:=sub}` — Assign is variable is null.
`${variable:?message}` — Abort if null.
`${variable:+sub}` — If variable is defined, return the substitution, otherwise, return null.
`${variable:start:length}` — Return a substring of `$variable`, starting at `start`, for a length of `length`.

In each case, if the variable is null or not defined, a substitution takes place. Take this command substitution as an example:

```
thelist=$(ls "${somedir}")
```

After executing this line, the variable $thelist will contain the output of the ls command for whatever $somedir is pointing to. However, if the directory that $somedir represents is empty, then $thelist will be null. Assuming that `ls $somedir` is null, then:

```
echo ${thelist:-empty}
```

will output "empty". The variable $thelist remains unmodified. Still assuming the $thelist is null, this substitution:

```
echo ${thelist:=empty}
```

will not only print "`empty`", but it will also *assign* the string "`empty`" (or whatever we supplied) to $thelist. A subsequent "`echo ${thelist}`" will now also print "empty".

The two substitutions shown make the substitution and let the script move along. The third version will exit after printing its message:

```
echo ${thelist:?"The list is empty"}
```

will print "`./scriptname: line x: thelist: The list is empty`".

To be clear, look at the short script in Listing 1.

Listing 1: test_script.sh

```
#!/bin/bash

echo ${thelist:?"The list is empty"}
echo "This is the last line."
```

This code will output "./test_script.sh: line 3: thelist: The list is empty" and the exit. The final line will never be executed. This is one way to ensure that an undefined variable doesn't cause havoc in a script.

The next one takes a little getting used to. The "+" bash string operator performs a test. If the variable is defined, return the substitution. If it is not defined (null), return null. This could easily be used to denote a Boolean. For example, if you just want to mark that an option is set, you could use the following code to set $debug to 1 if $debugFlag is not null:

```
debug=${debugFlag:+1}
```

Possibly the most common string operation is to extract a substring from a given string. You can use many tools external to bash to do this: cut, sed and awk immediately jump to mind. However, for simple substring needs, bash has a string operator for this built in. If $variable contains "The striped egg", then the following bash operation:

```
adj=${variable:4:7}
```

will assign "striped" to $adj. If the length is omitted, the substring runs from the start until the end of the string. Using the same string for $variable, this operation:

```
adj=${variable:4}
```

would assign "striped egg" to $adj. Interestingly, the colon operator has special interaction with $@ (the special variable containing all positional paramters). Instead of character positions in a string, the operator knows to act on each individual element. For example, look at the short bash script in Listing 2.

Listing 2: optest.sh

```
#!/bin/bash

echo ${@:3}
```

Running it with the following arguments yields the output shown.

```
$ ./optest.sh alfred bruce charles david
charles david
```

Bear in mind that I'm not suggesting you banish cut, sed and awk from your bash scripting. There are cases where they are

needed. Just don't overlook the substring function that's available within bash, that's all.

Pattern Matching

Typically, when looking to perform pattern matching on strings, bash users will call external utilities. However, bash has several pattern matching operators built in. Here's a list:

`${variable#pattern}` — Delete shortest match from beginning, return remainder.
`${variable##pattern}` — Delete longest match from beginning, return remainder.
`${variable%pattern}` — Delete shortest match from end, return remainder.
`${variable%%pattern}` — Delete longest match from end, return remainder.
`${variable/pattern/string}` — Replace first longest match of pattern in $variable with $string.
`${variable//pattern/string}` — Replace all matches of pattern in $variable with $string.

There are several external utilities that alter strings: sed, dirname and basename jump to mind. Their use can often be handled right within bash. This is a useful technique to use if possible, as it's more efficient to not have to fork off other processes.

Two months ago, I wrote about how I automatically populate a variable—$DB—with the path for my local Dropbox folder. Lets use the variable to pattern match on. Currently, $DB contains:

```
/Volumes/homes/Users/Shared/marczak/Dropbox
```

If we just want the path leading up to where the Dropbox folder is stored, we can quickly retrieve it with a pattern match. We know that we simply want to remove the "Dropbox" portion:

```
$ echo ${DB%%Dropbox}
/Volumes/homes/Users/Shared/marczak/
```

In fact, we can more generically replace the `dirname` utility: `${path_variable%/*}`

The opposite of dirname is basename, which returns only the file or directory name, void of the leading path. In our case, that's a little useless, since we know the name will always be "Dropbox". However, like the dirname replacement, we can make this a bit more general:

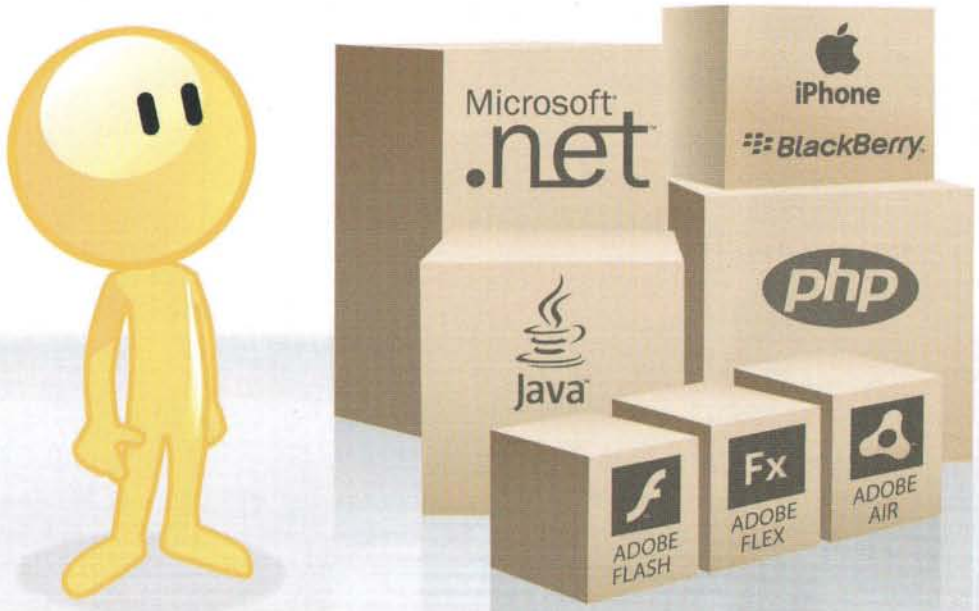
```
$ echo ${DB##*/}
Dropbox
```

The replacement operators allow for a lot of flexibility. These can replace most basic uses of sed. Let's look at a basic example:

```
$ name="Ed Marczak"
$ echo ${name/ */}
Ed
```

As I mentioned earlier, using the bash built-in can be more efficient. Now, if you have only one line in your script that calls an external utility, you're not likely to notice a performance benefit. However, what if that one line is called repeatedly?

NEED HELP TO IDENTIFY THE RIGHT TECHNOLOGY FOR YOUR BUSINESS?



YOUR SEARCH ENDS HERE!!

Transfer power to your business... Adopt Icreon's Technology Differential

Some of the services we offer

- Bespoke application development
- Web product development
- B2B systems integration
- Technical consultancy
- Content creation and management
- Systems migration
- Web marketing

Contact to inquire about technological solutions offered by Icreon:

Call: +1708.267.4825
Email: consulting@icreon.com
Visit: www.icreonglobal.com

Icreon
technology differential

I had the need to take a large list of filename pointers and strip a trailing character (" #$") from each. Used in a loop, the bash version beats using sed:

```
for filename in $(getnewfiles); do
  report(${filename/ #$/})
done
```

Many scripting languages have a **split** keyword (or equivalent), which returns individual elements of a single string separated into fields by a common character. In shell scripting, awk is often the natural place to turn. However, for simple splitting, you can stay right within bash. For example, the $PATH variable is a single string, with components separated by a colon character. If you were to replace the colon with a linefeed character, you could display or pipe each component into another utility as individual items. You could use awk to separate each field, or, a script like this:

```
IFS=':'
for i in $PATH; do
  echo $i
done
```

However, you can make this a clean, neat one-liner:

```
echo -e ${PATH//:/'\n'}
```

The string pattern substitution replaces ever occurrence of a colon with a '\n' string (note the double forward-slash characters). The -e flag to echo forces it to treat '\n' as a linefeed character, giving this one-liner the same behavior as the loop shown just above (where, in all honesty, you should also save $IFS prior to the loop, and then restore it afterwards).

Just remember these operators and try them before shelling out to sed or other utility.

Length Operator

The last string operator we'll cover is the length operator. This is a natural replacement for shelling out to "wc -c" (count characters). If you need to obtain the length of a string or list, the length operator may just solve your needs.

${#variable} — return the length of the variable as a string.

You may need to test the length of a string for some purpose. For example:

```
if [[ ${#1} -gt 10 ]]; then
  echo "String too long!"
fi
```

If the first parameter passed into the script or function is greater than 10 characters, the conditional will succeed and print "String too long!" Nicely, this also works with lists, such as $@:

```
if [[ ${#@} -lt 3 ]]; then
  echo "Too few parameters!"
fi
```

This snippet of code complains if there are less than three parameters passed into the script or function that it is contained in.

Conclusion

bash has some great functionality that is often overlooked. Additionally, calling external utility programs impacts performance in terms of speed and resources. While it isn't always a dire need, there are many cases where you want to be mindful of resources used (disk, CPU, thread count, etc.)—a shared server, for instance.

Media of the month: "Results Without Authority" by Tom Kendrick. I've mentioned project management books in the past, and this is another gem. We all lead projects in some manner, however, few of us have formal training. While books alone aren't a complete replacement for training (and actual *doing*), they're a good start and can plant ideas for you to try. "Results Without Authority" probably applies to many of us: how to lead a project when you're not directly a manager over many of the people involved in getting the project to completion.

I hope you're gearing up for the first MacTech Conference: http://www.mactech.com/conference. Join us in Los Angeles for three days of learning and interaction with your peers. I'm thrilled that this is taking place and hope you are too! I hope to see you in L.A.!

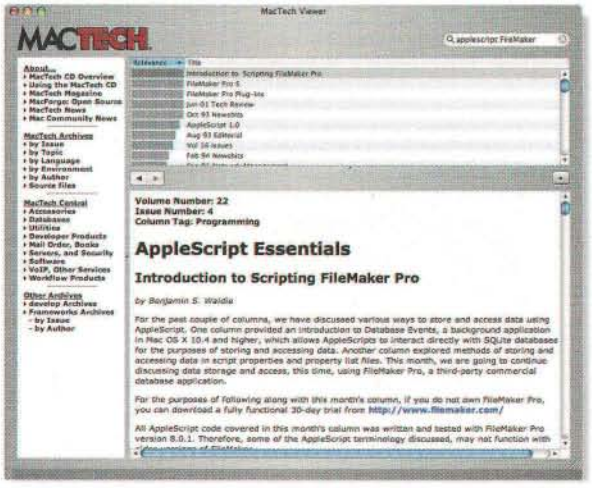
About The Author

Ed Marczak is the Executive Editor for MacTech Magazine, and has written the Mac in the Shell column since 2004.

The MacTech DVD - Volumes 1.01-25.12 is packed with more than ever before -- over 3200 articles from more than 290 issues **(1984 - 2009)** written by over 870 experts, all 29 issues of Apple's develop, 21 issues of FrameWorks magazine, 100+ MB of source code, MacTech Viewer, working applications, full documentation, demos for techs, ***and more!***

Everything is displayed in the very fast, very searchable ***MacTech Viewer!*** An application that has been designed specifically with Techs in mind. Search quickly through 25 years of great information provided by MacTech. Information to save you time, and make your life easier.

Requires Mac OS X v. 10.4.5 or later



Toll Free 877-MACTECH, Outside US/Canada: 805-494-9797 • http://www.mactech.com/dvd/

CoreSec

Security topics for administrators and programmers

By Michele (Mike) Hjörleifsson

Welcome

Welcome to the first installment of CoreSec, a continuing series of articles to answer questions, address common concerns and introduce security technologies and concepts. In the following months, we will delve into interviews, topical discussions, how-to articles, and Q&A from you, the reader. Why the name "CoreSec?" At its core, OS X is UNIX 3 compliant. Its core components all address security in one form or another. Whether it's protected memory, secured virtual memory or email encryption, security runs through the core of OS X and hence the name "CoreSec."

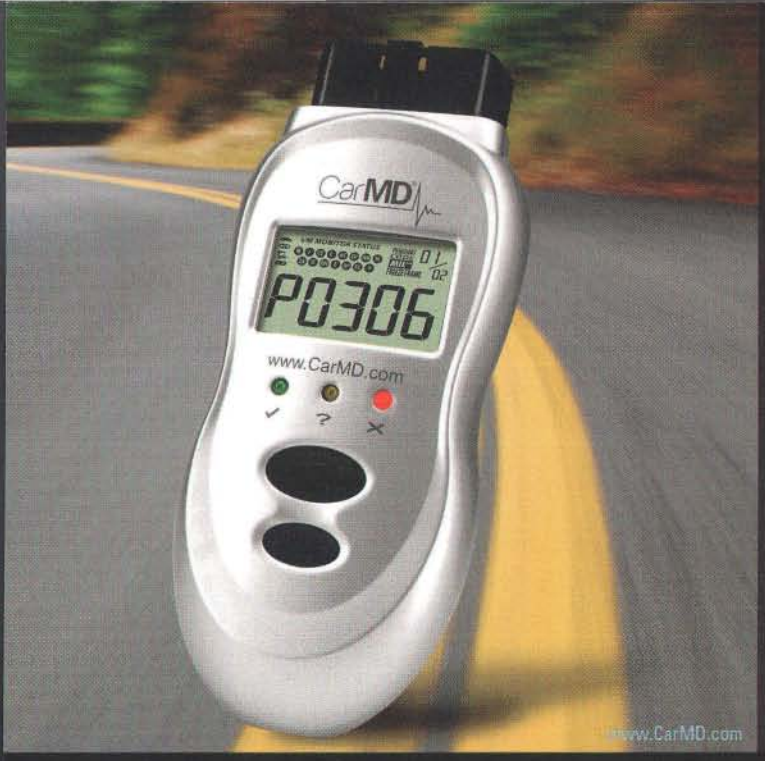
Why

The term "security guy" was akin to the term "computer guy" ten or fifteen years ago. A "security guy" would be expected to be the point person on all things security related, but like the "PC guy" or "Apple guy" of years ago, the security technologies requirements, mandates and legislation have branched the security industry and technologies into many different areas that today seem to touch all of us regardless of our position, industry or title. For instance, a movie editor may have security concerns about leakage of confidential information on their movie or corporate video project. Doctors are all too familiar with the acronym HIPPA and the related protection of patient information. Banking, securities and stock brokerages deal with a myriad of security concerns and legislative requirements. Accounting firms must ensure the confidentiality and security of their customers' data and subsequent transmission of that data. Last but certainly not least, any organization accepting a credit card for their brick and mortar or web-based transactions have a host of regulatory requirements they may not even be aware of until it is too late. The goal of this column is to provide information, links to resources and discussions to aid you in your day-to-day life addressing security tasks and concerns.

Let's start with a set of regulations you may or may not be aware of but are subject to if you engage in acceptance of any type of credit card transactions. This literally means any acceptance of ANY transaction regardless of whether it is once a year or many thousands of times a day.

PCI (payment card industry) regulations (set by the PCI Security and Standards Council) are a set of rules and requirements enforced on all merchants, processors and credit card companies. PCI was initially established to avoid government intervention in the credit card industry. The idea behind PCI is that the industry can provide its own set of regulatory rules and requirements robust enough to protect consumer information; credit card and personal data and no governmental regulation or laws would be required. The outcome of that premise is yet to be seen but if you accept credit cards in any form (via web, phone, or in person) your organization is subject to these regulations and the subsequent penalties for ignoring or violating the requirements. As you may have never heard of PCI, it was established in 2006 and from their website their charter states, "The PCI Security Standards Council is an open global forum, launched in 2006, that is responsible for the development, management, education, and awareness of the PCI Security Standards, including: the Data Security Standard (DSS), Payment Application Data Security Standard (PA-DSS), and Pin-Entry Device (PED) Requirements." Wow, a lot more acronyms to memorize and confusing industry terms from the credit card industry. What does it all mean? Well, simply put, they are the credit card police, providing and enforcing rules on merchants, software manufacturers, processor and even hardware manufacturers that work with credit cards.

Since 2006 the PCI-SSC has set and modified its rules and regulations to address growing concerns about data loss, identity theft and credit card fraud and has added serious teeth to its rules and regulations to ensure compliance. Knowledge, or lack thereof, of the rules is not an excuse for non-compliance and will not excuse an organization from the penalties enforced if the regulations are not complied with. For the smallest merchant (one accepting less than 20,000 in credit card transactions per year), fines can range from $10,000 to a maximum of $150,000. So let's take a look at some of the basic rules that apply to you, the retail or web merchant. DSS (Data Security Standards) layout the



1 Plug It In!

Use from the safety and comfort of your front seat. Works on all 1996 & newer cars, light trucks, SUVs and minivans.

2 Get The Info!

Everything OK
Possible Problem
Service Required

CarMD makes it easy to quickly identify your vehicle's health.

3 Run A Report!

CarMD.com provides you with empowering information. Walk into any repair shop knowing what needs to be fixed, what doesn't and how much it should cost.

WITH CarMD®, CATCHING HIDDEN ENGINE PROBLEMS IS EASY AS 1-2-3

When you have an important deadline at work or a weekend getaway planned, the last thing you need is for car trouble to slow you down.

In today's cars and trucks, virtually everything's computerized. And even if you do have a degree in Computer Science, it's hard to determine whether that dashboard-warning light is serious or simple. Thanks to the revolutionary CarMD® handheld device and software kit, now virtually everyone who owns a car and a computer can quickly and easily solve automotive repair mysteries.

How it Works

CarMD gives the average driver a way to catch hidden engine problems before the dashboard warning lights come on and they become expensive repairs. Just plug the handheld CarMD device into your vehicle's Data Link Connector (DLC). The DLC is a small port found right under the dashboard on 1996 and newer vehicles – even the newer model hybrids. It's the same place your mechanic plugs in his expensive diagnostic tools. Not sure where to connect? Visit www.CarMD.com to search by year, make and model.

In seconds, CarMD beeps to confirm the test is complete. Then the tool's built-in LEDs let you know how severe the problem is (Green = OK, yellow = possible problem, red = service required). These handy lights are a great litmus test when you're shopping for a used vehicle, or getting ready to hit the road for a long getaway.

To learn more about your car's problem, connect CarMD to your computer using the included software and USB cable. CarMD customers gain free access to an extensive online database (www.carmd.com) that helps diagnose the most likely fix and estimate what repairs should cost down to fair parts and labor in your region.

Coverage

CarMD works on 1996 and newer cars, light trucks, minivans and SUVs – foreign and domestic. It monitors everything that's covered by a vehicle's on-board diagnostic system, which translates to about 80 percent of the systems ranging from a loose gas cap to a catalytic converter failure. Recently launched CarMD Version 3.0 software, which is now compatible to run natively under Mac OS X in both Power PC- and Intel-based Macintosh computers (Version 10.4.4 and newer– including Snow Leopard). Lifetime software and firmware updates mean the CarMD tool you buy today will work on the car or truck you invest in tomorrow.

Special Offer* $88.99

(a $10 savings)

When you buy online at: www.CarMD.com

Enter promo code MacTech2 at checkout

*Offer good through May 31, 2010. Act now!

CarMD
FOR THE LIFE OF YOUR CAR

Mac Universal

STAY GREEN

CarMD can help keep your car – and your wallet – greener too. The "Check Engine" light often indicates problems with the emissions system. CarMD can diagnose problems that lower your mileage and pollute the environment. Its color-coded LEDs will also tell you if your vehicle is ready to pass its emissions test, so you don't take time off work for a smog check you might fail. CarMD customers gain access to staying healthy reports that include technical service bulletins and safety recalls that can often lead to free repairs.

> **"If I did not have the CarMD analyzer and computer software I would have wasted approximately 3 hours going to the dealership. As far as I'm concerned, it has paid for itself already."**
>
> *– RobinJ., N. Ridgeville, OH*

To learn more, visit CarMD at:

www.CarMD.com

requirements for you, the web or retail merchant. This will also include consultants, doctors, dentists and anyone who accepts a credit card for payment. The requirements are split into six main categories with up to three topics per requirement. The details on the level of protection mandated for each of these requirements depends on the number of transactions and the position in which your organization resides in the credit card processing chain. For instance, a credit card processor has much more stringent requirements than a web store or doctor's office.

Many organizations I speak with about security aren't even aware that they are subject to these rules, regulations and requirements and have been accepting credit cards for years. Fortunately, OS X has many built-in tools to help you meet the requirements. Let's take a look at the DSS categories, topics and explore how OS X can help you address them. Note that although they are not discussed in detail, each topic has specific requirements.

Category One: Build and Maintain a Secure Network

Topic One: Install and maintain a firewall configuration to protect cardholder data

Topic Two: Do not use vendor-supplied defaults for system passwords & other security parameters

OS X provides a built in firewall both on the standard desktop operating system as well as a more granular controlled firewall for OS X server, this can easily meet the first requirement. Password policies enforced by Open Directory on OS X server can meet the second requirement and enforce strong password and password renewal policies.

Category Two: Protect Cardholder Data

Topic One: Protect stored cardholder data

Topic Two: Encrypt transmission of cardholder data across open, public networks

The first topic really depends on whether you even store cardholder data and the application used to store it. For instance, if you use a service such as authorize.net you should never have to store cardholder data so this topic does not apply but the second topic would and can be addressed easily by ensuring that you are transmitting this data over SSL and receiving the data from your clients from your web applications on an SSL-secured webpage/website. OS X provides a tool called Certificate Assistant (launched from Key Chain Access located in your Applications > Utilities folder) that can request and install SSL certificates for your website.

Maintain a Vulnerability Management Program

Topic One: Use and regularly update anti-virus software

Topic Two: Develop and maintain secure systems and applications

The first topic, while widely considered unnecessary on OS X systems, is a good precaution and on OS X server there is a free anti-virus engine included for desktop systems. You can download iAntiVirus, Norton's AntiVirus or a host of other commercially available tools. Alternatively, you can download and install

APPRIVER CLIENTS NEVER SEE THIS

Error: Mailbox Full

You have exceeded the storage limit on your mailbox. Delete some important emails or contact your server administrator who is on vacation right now. My neighbor's kid is really good with computers. Maybe he could help you out when he gets home from school. Or call AppRiver to solve this problem now.

Call AppRiver

What's New:

Native support for Microsoft® Exchange allows users to integrate their Exchange data into Mac's **Mail, iCal** and **Address Book** applications.

Unlimited Mailbox Storage
No need to manage your mailbox storage quotas.

Spam & Virus Protection
Comprehensive protection against unwanted email.

Entourage & Mac Mail Optimized
Email Performance you would expect on your Mac!

Better Mobility
Synchronize your iPhone with our FREE ActiveSync service, the fastest, most reliable synchronization service available.

appriver®
The Messaging Experts™

www.appriver.com/mac

sales@appriver.com

(866) 223-4645

ClamXav: an open source antivirus application from http://www.clamxav.com If your organization develops their own web applications, desktop applications or web store you must ensure that the latest security patches are installed, that you are using encryption such as SSL to encrypt and protect cardholder data. Since these requirements are very specific to the programming technology you are employing we will have to leave this to your application developer to research.

Implement Strong Access Control Measures

Topic One: Restrict access to cardholder data by business need-to-know

Topic Two: Assign a unique ID to each person with computer access

Topic Three: Restrict physical access to cardholder data

Topic One is more of a business process than a computer security item but ACLs (access control lists) can be used to control access to documents and applications containing any cardholder data. Additionally, most accounting and card processing applications have role-based authentication, providing you with granular control over who has access to this data. The same techniques can be applied to meet the second topic's requirements. The third topic is specifically a business process, though the Mac computers that store this data can be set up with firmware passwords or security devices such as smart-cards to restrict physical access to the Mac computers storing the cardholder data.

Regularly Monitor and Test Networks

Topic One: Track and monitor all access to network resources and cardholder data

Topic Two: Regularly test security systems and processes

OS X provides robust logging options for its firewall, file access and network access, which can be an effective way to meet the requirements of the first topic. The second topic is more procedural; a policy and test plan should be implemented by your organization to ensure compliance with all the categories and associated topics/requirements.

Maintain an Information Security Policy

Topic One: Maintain a policy that addresses information security

The last category and topic are addressed at the organizational owner or responsible party and mandate the documentation of a security policy and subsequent reviews of the policy to ensure that it effectively addresses the compliance with the aforementioned categories and their associated topics and regulations. For more information on PCI DSS you can visit https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml which contains the regulations, requirements, and dozens of educational resources.

Conclusion

As witnessed by the PCI discussion, something as simple as accepting a credit card for payment, donations or surety of payment has security implications that need to be addressed by your organization. It is my goal in this monthly column to make you aware of issues such as these and help you address them in your organization. We strive to make this column as pertinent to you, the reader, as possible so please feel free to email mikeh@mactech.com and provide feedback or topics you would like to see addressed in our upcoming editions. Stay safe; stay secure.

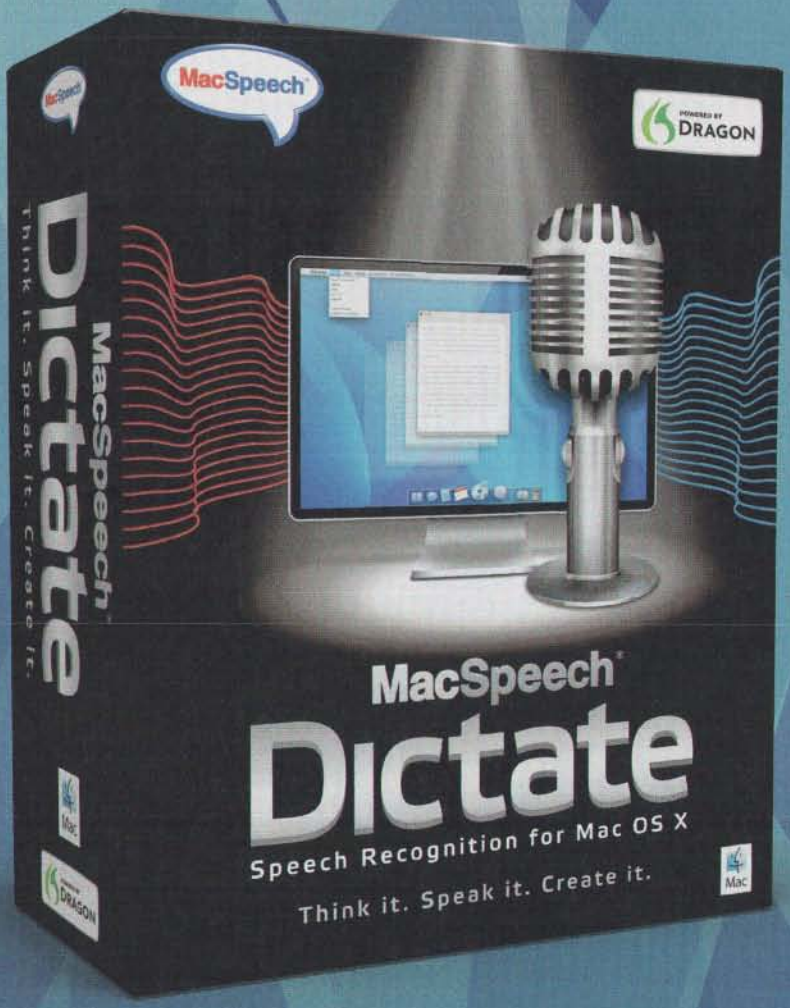
About The Author

Michele (Mike) Hjörleifsson, co-author of the* Apple Training Series: Security and Mobility *courseware has been developing on the Apple platforms since the Apple][+, implementing network and remote access security technologies since the early '90s, and worked with the nation's largest corporations and government institutions, authoring white-papers, technical magazine articles and topical discussions at IETF (Internet Engineering Task Force), and other organizations on security topics, and podcasting with Apple Podcast Producer. He is currently working with companies worldwide on Apple and Security consulting projects and conducting Apple IT and Pro Apps training. Feel free to contact him at mhjorleifsson@me.com

You talk. It types.

MacSpeech Dictate

Premier speech recognition for the Mac.



www.macspeech.com

Available from MacSpeech, Apple, and other fine Macintosh retailers.
Visit the MacSpeech website for a complete retailer listing.

MacSpeech

~~Type~~. Talk!

POWERED BY DRAGON

iPad in the Enterprise

Is Apple's latest creation an enterprise tool?

By Greg Neagle, MacEnterprise.org

MacEnterprise.org
Mac OS X enterprise deployment project

Introduction

Apple's iPad shipped in the United States on April 3rd, 2010. Like the iPhone before it, even though it was not specifically designed for use in an enterprise setting, it will find its way into large organizations.

The first time my daughters saw my new iPad for the first time, they each exclaimed, "It's a giant iPhone!" And of course, from one perspective, it is (or probably more accurately, a giant iPod Touch). Since the iPhone is being used successfully in enterprise environments, it stands to reason that the iPad will have its uses as well.

So let's quickly review the enterprise-friendly iPhone/iPod Touch features that carry over to the iPad.

Email/Calendar/Contacts

iPad's email client can connect to enterprise email systems; both those based on standard IMAP/SMTP services as well as Microsoft's Exchange via ActiveSync. Exchange calendars and contacts can also be synced with an iPad. If your organization's calendar supports WebDAV or the iCal standard, you can at least get a read-only version of your work calendar on the iPad.

Because of the extra screen space, iPad's email client is more capable and easier to work with than the iPhone version. The same mostly applies to iPad's calendar application as well.

Web access

Safari on the iPhone raised the bar for the mobile web. With the exception of Flash, for the first time you could use "real" web pages on a mobile device. iPad's Safari is faster and makes great use of the added screen space. If you've been able to use your organization's web applications on the iPhone, the iPad will work even better.

802.1X and VPN

Like the iPhone, the iPad supports the enterprise 802.1X Wi-Fi standard and many VPN implementations. I had no trouble connecting my iPad to my organization's secure Wi-Fi network. I was unable to test the VPN due to policy restrictions in my organization.

Configuration profiles

With iPhone OS 2.0, Apple introduced the iPhone Configuration Utility, which allowed administrators to create configuration profiles for the iPhone (and iPod Touch). When these profiles are installed on an iPhone, they can configure the device for email access, calendar access, 802.1X, VPN, and more. In my testing, existing configuration profiles generated for the iPhone/iPod Touch worked as expected on the iPad. This allowed fast, secure configuration of my iPad to connect to my organization's email, calendar, and secure wireless network.

The iPad shares some of the iPhone's shortcomings for enterprise use. For example, the initial setup requires connecting to iTunes on a Mac or PC. This is workable for personal use, but if you need to setup and configure a large number of iPads in an enterprise environment it seems like an annoying extra step.

Laptop Replacement?

So you've probably already figured out that the iPad is more capable and at least as useful as an iPhone or iPod Touch in an enterprise setting, but what you're really wondering is – can an iPad possibly be used as a laptop or even desktop replacement for some users?

Let's look at a few things the iPad can do that the iPhone (currently) can't.

Display

First: the screen size. The iPad's 9.7-inch diagonal, 1024x768 display is closer in size to a laptop or desktop display (and the same number of pixels as many 15-inch computer displays a few years ago). The extra screen real estate makes you more productive in mail and web browsing. It also makes remote access applications like VNC, RDP, and SSH applications much more usable to connect to, view, and control remote machines, making it far more practical to use an iPad, rather than an iPhone/iPod for systems administration tasks. Other classes of applications can present much more information at once, again making you more productive.

Hardware Keyboard

Second: you can use a hardware keyboard with the iPad. Apple offers a keyboard dock, which integrates a keyboard with a 30-pin connector dock. Since the 30-pin connector is on the "bottom" of the iPad, that limits its use to portrait orientation. If you want or need to use the iPad in landscape orientation, the keyboard dock is of no use. And you'll almost certainly want to change the orientation of the iPad from time to time – some apps, like Keynote, work only in landscape orientation; others, like Pages, behave differently depending on the orientation. Pages hides all of its controls in landscape mode. If you want to format your text or insert pictures, you need to turn the display to portrait mode to get the controls to appear.

Using a Bluetooth keyboard might be a better choice, as it can be used with any iPad orientation. Apple offers a compact Bluetooth keyboard, but any Bluetooth-compatible keyboard should work.

Using a hardware keyboard transforms the iPad from being a "giant iPod Touch" into an almost-laptop – you can imagine yourself using it for more than mail and web browsing. Some Mac keyboard shortcuts work as you'd expect – the key combinations for Copy, Cut, Paste, Select All and Undo all work. Others, like the traditional key combinations to format text as bold, italic, underlined, etc, seem to be unsupported.

iWork apps

Third: the iPad can run some Office-like apps: Apple offers iPad versions of Pages, a word processor; Numbers, a spreadsheet, and Keynote presentation software. If you are a Mac user, you probably have at least heard of the Mac versions of these applications. Like their Mac counterparts, the iPad versions can import and export Microsoft Office documents as well as work with their own document formats.

VGA output

Fourth: with Apple's iPad-to-VGA adapter, you can connect your iPad to most business projectors and use the iPad for presentations. When you playback a Keynote presentation with the adapter attached, the presentation plays back on the attached monitor or projector, and the iPad display is used for controlling playback and a virtual "laser pointer."

Figure 1 – Keynote's controls when presenting to an external display

Enterprise Shortcomings

So, back to the question at hand: can an iPad possibly be used as a laptop or even desktop replacement for some users?

Sadly, I'd have to say the answer for most users is "not yet," especially if you need to do document creation or editing. There are just too many issues, most involving workflow.

Printing

Even though the "paperless office" is the future, the future isn't here yet. Enterprise works still need to create paper documents. Unfortunately, iPad has no integrated printing mechanism. You can't print from the mail application, Safari, or the iWork apps. Apple suggests you transfer the documents to a Mac or PC and print from there. The App Store offers a few apps for printing, but none are integrated with the OS.

File transfer

Document transfer is annoying and clunky and painful. Apple suggests you use iTunes to transfer documents to and from the iPad. This, of course, requires you to connect the iPad to a Mac or PC with a cable and use the iTunes interface to choose documents you want to transfer. Again, this might be an acceptable workflow for home use, where you just want to move some documents between your personal Mac and your personal iPad. But if you want to grab a copy of a document from a colleague, having to connect your iPad to their Mac just seems wrong.

You can also transfer documents via email, and if they are a format that one of the iWork apps can import, when you preview the document in the Mail application, you'll see a button to open the document in the relevant iWork app. This, right now, is probably the best choice for ad-hoc document transfer, because it can be done wirelessly and requires no special setup.

The final suggestion from Apple is to use the public beta of its iWork.com service, which allows you to publish and share iWork documents via the web. I was able to publish an iPad-edited document on iWork.com, and import another document that I had published from Pages on a Mac. This workflow means, though, that you could have three or more copies of the document to reconcile - a copy on a Mac, the copy hosted at iWork.com, and the copy on the iPad. Keeping all this straight seems error-prone.

Apple's iDisk – part of the MobileMe services – seems like it might be another option. As of this writing, Apple had not yet updated its iDisk app for iPad; perhaps in the future one might be able to use iDisk for document transfer.

Document conversions

Assuming you've found some way to get a document onto the iPad, and you've purchased the relevant iWork app, your challenges are not at an end. If your documents originated in Microsoft Office, you may be able to open them in Pages or Numbers or Keynote, but not with 100% fidelity. In Pages, some Word document formatting will be lost, fonts will be substituted, and more advanced features like "Track Changes" will be turned off. Numbers likewise supports only a subset of Excel features. Imagine a workflow in which you worked on a document in Microsoft Word on a desktop

computer, emailed it to your iPad, imported it into Pages, worked on it, exported it back to a Word document, emailed it back to your desktop computer, and re-opened it in Microsoft Word. That workflow was painful to write; I'd imagine it would be much worse to actually do on a day-to-day basis.

If you think the main pain is caused by the Office-to-iWork-and-back conversions, and your problems will be solved if you use iWork '09 on the desktop as well as the iPad, you'll be disappointed.

iPad's iWork applications, though impressive as a rethinking for the multitouch OS on the iPad, are still essentially "lite" versions of the corresponding Mac desktop applications. All three applications support a subset of the fonts available on the desktop, and each app drops some features of its desktop version.

Let's consider Keynote as an example. While it would be tempting to hope you could take an existing Keynote presentation, transfer to your iPad, hit the road and present it from your iPad, in practice you'll almost certainly have to make some changes to the imported document so it will display acceptably. When importing a fairly simple Keynote presentation I created over a year ago, I found the iPad version couldn't display one of the graphics on a slide and substituted the Monaco font I had used for shell scripting examples on several other slide with the Helvetica font, making the examples hard to distinguish from other text. Both problems were easy enough to fix, but the point remains that Keynote for iPad is not 100% compatible with Keynote for Mac OS X.

If you use more advanced Keynote features, you may find they also are not supported on the iPad, leaving your presentation almost unrecognizable. Not all transitions, animations, and chart types are supported. Tables may be reformatted. Early reviewers have noted that not only does Keynote for iPad not support speaker's notes, it actually deletes the notes on import. This makes creating a presentation on a Mac, transferring to an iPad for tweaking, and transferring back a dangerous operation. Oddly, when using Keynote with the iPad-to-VGA adapter, there seems to be plenty of room for speaker's notes on the iPad's display – even with the on-screen playback controller, and displaying speaker's notes doesn't seem computationally complex, so this is a curious omission.

Working with iWork

You definitely can get more "business" use out of an iPad than an iPhone; but in most cases you'll still need a laptop or desktop computer available to you to finish a job 100%. For example, I used an iPad with the Apple Bluetooth Keyboard to type the first draft of this column in Pages. I later emailed the document to myself and copied and pasted the text into a Word template for this column, where I finished the formatting and editing.

There are elements to using the iPad for this sort of work that take a little getting used to. In Pages for iPad, it's a bit liberating to not have to worry about saving the document. Pages on the iPad automatically saves your work, as do Numbers and Keynote. Hitting the home button, which quits Pages so you can open a different app, like Mail, is initially a bit disconcerting. But switching to another application is fast, and when you return, Pages *almost* remembers where you left off (the text is in the same position, but

/n software

www.nsoftware.com >

The Best-Selling Internet Communications, Security, and E-Business Components, Now Available On:

Mac OS X and iPhone!

COCOA FRAMEWORKS FOR INTERNET COMMUNICATIONS

IP*Works! eliminates the complexity of Internet development providing easy-to-use, programmable components that facilitate tasks such as sending E-mail, transferring files, managing networks, browsing the web, consuming Web Services, etc.

Internet Communications

- **IP*Works! - [Core Framework]**

A comprehensive framework for Internet development. The core building block for most /n software products.

Components - IPMonitor, MX, REST, NetCode, RSS, NNTP, SMPP, POP, Rexec, Rshell, Syslog, SMTP, WebDav, SOAP, XMPP, Telnet, Ping, TFTP, FilerMailer, UDPPort, HTMLMailer, WebForm, NetClock, WebUpload, RCP, Whois, FTP, XMLp, HTTP, SNPP, IMAP, MIME, IPInfo, IPDaemon, IPPort, NetDial, LDAP, ICMPPort, MCast, TraceRoute

Network Management

- **IP*Works! Secure SNMP**

A comprehensive toolkit for building Secure SNMP-based agent and manager applications including advanced SNMPv3 security features, trap handling, and ASN-1 MIB compilation.

Components - SNMPAgent, SNMPMgr, SNMPTrapMgr, MibBrowser

File & Streaming Compression

- **IP*Works! Zip**

Suite of easy-to-use, fast, effective components for compression and decompression with advanced features including self-extracting archives, industrial strength AES encryption, and Zip64 archives.

Components - GZip, Tar, Jar, Zip, ZipStream

Secure Connectivity & E-mail Confidentiality

- **IP*Works! SSL**

SSL-enabled versions of the components in the core IP*Works! package.

Components - WebFormS, LDAPS, WebUploadS, RESTS, NNTPS, IPDaemonS, POPS, FileMailerS, SOAPS, HTMLMailerS, TelnetS, HTTPS, FTPS, IMAPS, SMTPS, IPPortS, CertMgr, RSSS, SMPPS, WebDavS, XMPPS

- **IP*Works! S/MIME**

Components for E-mail and file confidentiality, authentication, and non-repudiation through encryption and digital signatures. Implements the S/MIME standard for digital security.

Components - CertMgr, SNNTP, SPOP, SIMAP, SSMTP, SMIME, SFileMailer

- **IP*Works! SSH**

Secure Shell (SSH) enabled client communications components supporting strong encryption and advanced cryptography. A highly-evolved code base, enhanced with enterprise features like IPv6 addressing and 64-bit support.

Components - SFTP, SShell, SExec, SSHClient, SSHTunnel, CertMgr

COCOA FRAMEWORKS FOR INTERNET BUSINESS

Built using the same technologies as our award-winning IP*Works! product line, these packages offer native components for Credit Card Processing, Online Payments, E-Banking, Shipping & Tracking, and more!

- **QuickBooks Integrator**
- **E-Payment Integrator**
- **E-Banking Integrator**
- **TSYS Integrator**
- **Paymentech Integrator**
- **FDMS Integrator**
- **FedEx Integrator**
- **USP Integrator**
- **USPS Integrator**
- **PayPal Integrator**
- **Amazon Integrator**
- **SharePoint Integrator**

GET THEM ALL THROUGH OUR COMPREHENSIVE RED CARPET SUBSCRIPTION, $1,499 PER DEVELOPER, ROYALTY-FREE. 20% DISCOUNT FOR MACTECH SUBSCRIBERS.

the insertion point is lost). It's almost as nice as "real" multitasking. When iPhone OS 4.x makes it to the iPad in the fall, presumably the experience of switching to other apps and switching back will be even nicer, but it's acceptable now.

Unlearning over two decades of the mouse, menu and keyboard interface paradigms is another challenge, and since I can't yet give up my laptop, I can't really unlearn; instead I have to switch modes in my brain. I keep reaching for a mouse to move the insertion point to make a selection, and missing a menu bar. I think the more direct manipulation model in the iPad is the future, though, and as I get used to the iWork for iPad user interface, I will become more proficient. (As I write this, it's been less than a month since the iPad hit store shelves in the US!)

Still, in my short time with the iPad and the iWork apps, I can imagine doing light work in Pages and tweaking and giving presentations in Keynote. I find it harder to imagine doing any spreadsheet work at all in Numbers – I think it's the app that suffers most from throwing out the menu/mouse/keyboard interface. Working in Numbers feels like working on a spreadsheet with giant oven mitts on your hands.

Some notes on the iWork apps:

Pages

If all you need to do is write text, using Pages with a hardware keyboard works well. With the iPad in landscape orientation, all the user interface controls go away, leaving a blank, clutter-free page for writing. Rotate the iPad to portrait orientation, and controls for formatting, inserting pictures, charts and graphics appear.

Figure 2 – Pages formatting controls

Numbers

This is the app where the changes in user interface make for the steepest learning curve. I tried to make a simple spreadsheet that replicated a multiplication table. I was successful, but it took me almost half an hour. The same task on a laptop or desktop would have taken me around a minute. This is not to say that a simple spreadsheet would always take so long; I'm sure as I got used to the interface I could work more quickly. (And to be fair, I have never used Numbers on the Mac – all of my spreadsheet work has been with Excel.) But it does underline the fact that iPad is a totally new platform – in many ways it's harder to move from Mac to iPad than from Windows to Mac. Windows and Mac OS X share many interface conventions; iPad throws out most of them. Expect it to take time to attain productivity in iWork apps for iPad.

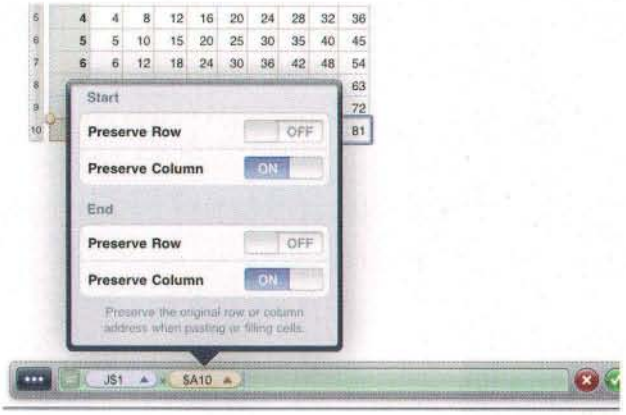


Figure 3 – Editing a Numbers document

Keynote

Since a Keynote document typically consists of slides with small amounts of text and large graphic elements, it's probably the best suited to the direct manipulation model of iPad OS and the iPad iWork apps. I found it pretty easy to manipulate an existing presentation. Keynote works only in the landscape orientation, but that wasn't a real problem. I can actually imagine using this application without a hardware keyboard.

Figure 4 – Editing a Keynote presentation on the iPad

What about iPhone OS 4?

Five days after the iPad shipped, Apple announced iPhone OS 4 would ship in the summer, and would come to iPad in the fall. Of the details shared so far, there are a few that would be useful to enterprise iPad users.

Multitasking

The new multitasking features in iPhone OS 4 will definitely help users be more productive on the iPad. While switching between apps is pretty fast now, apps don't always store their state with 100% fidelity, so when they relaunch, you may have to take some action to get back to what you were doing when you

switched away from the app. The new fast app switching and better saving of application state will make it much faster to switch from Pages to email and back without losing your place.

Enterprise features

Apple also announced more enterprise-friendly features, but did not go into much detail yet. There are new data protection APIs to allow apps to encrypt private data. New support for "Mobile Device Management" will allow enterprises to more easily deploy, configure, manage and update iPhone OS devices. Enterprises will be able to deploy internal apps wirelessly – no more needing to connect the device to iTunes to download internal apps. And the improvements in mail allow multiple Exchange ActiveSync accounts, a unified inbox, and organization of messages by conversation threads.

Maybe more?

Since the iPad is not getting the iPhone OS 4 update until later in the fall, hopefully this is a sign that a few more iPad-specific features are on their way. If there are additional features, maybe some will address a few of the enterprise workflow issues (file transfer, printing, document compatibility) we've looked at here.

Don't forget that Apple and third parties can also update the iPad experience through the release of new and updated applications via the App Store. We may not have to wait until the fall for updates of interest to enterprise.

Conclusion

As of this writing, it's been less than a month since Apple has released the iPad. In many ways, iPad is a completely new platform, even more different from the Mac than the Mac is from Windows. While Apple didn't design the iPad with enterprises in mind as a primary customer, it's not too much of a stretch to predict that iPad will find its way into large organizations as well as the iPhone has already done.

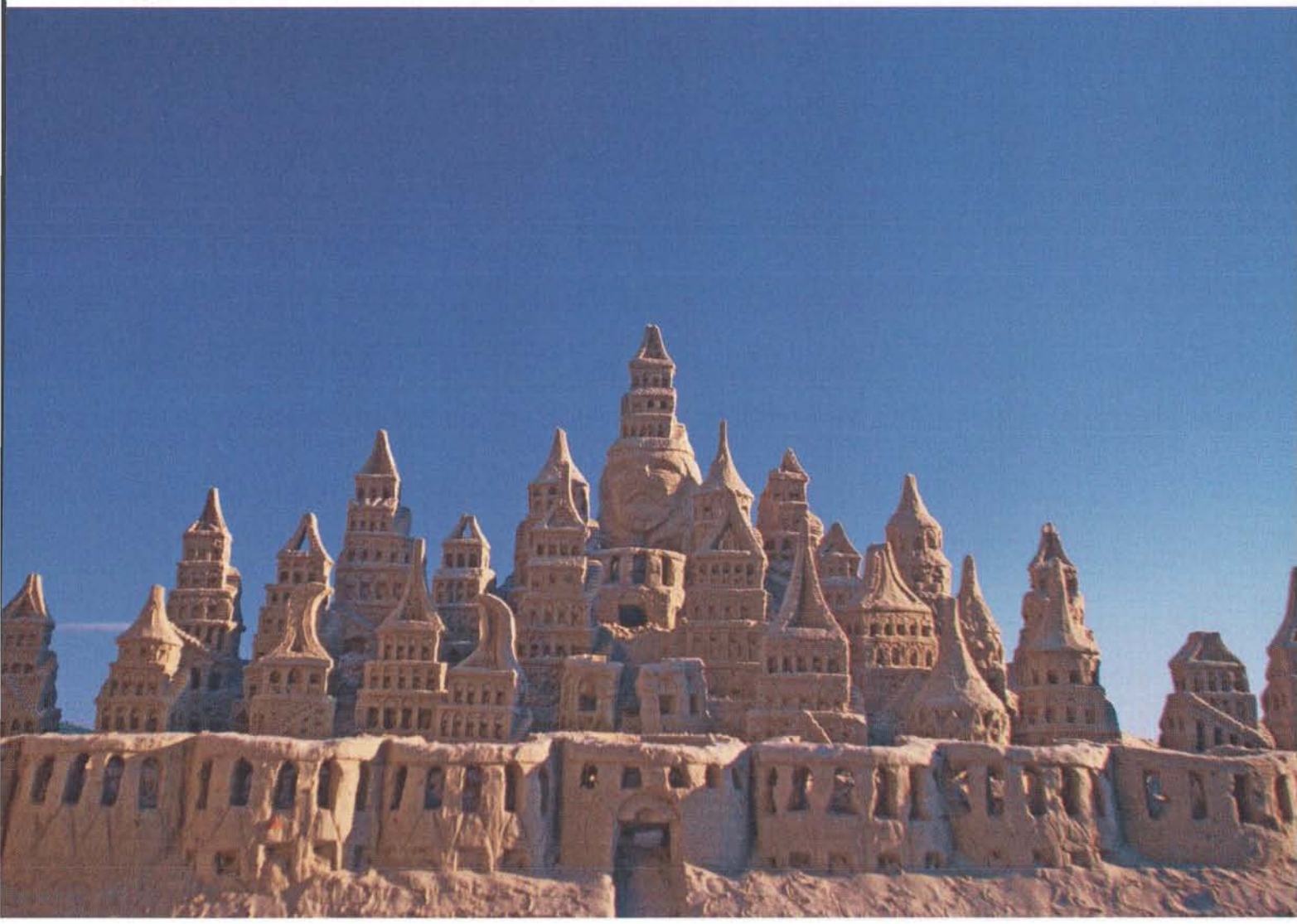
I expect this new platform to evolve and grow over time, just as the Mac has evolved from its beginnings on a machine with a 1 MHz processor, 128K of RAM, and a nine-inch black-and-white display. Whether the iPad platform in its current state is useful in your enterprise is ultimately up to you to decide. Even if the iPad probably can't serve as a laptop or desktop replacement machine right now for most enterprise users, you owe it to yourself to keep an eye on this platform, as there will be many more devices like this in the future, and their capabilities will only continue to grow.

MT

About The Author

Greg Neagle is a member of the steering committee of the Mac OS X Enterprise Project (macenterprise.org) and is a senior systems engineer at a large animation studio. Greg has been working with the Mac since 1984, and with OS X since its release. He can be reached at gregneagle@mac.com.

imagine. amaze. inspire.



you think it. you create it.

www.realsoftware.com

REAL Studio is the powerful, easy-to-use tool for creating your own software. At REAL Software, we call it a problem solver. You've probably said, "Wouldn't it be great if there was an application that ..." REAL Studio fills that blank.

Download your free 30-day trial edition today! Or buy now - REAL Studio comes with a 90-day money back guarantee.

REAL Studio compiles native applications for Mac OS X, Windows and Linux from one set of source code. Each version of your software looks and works just as it should in each environment. You can even create a Universal Binary with one mouse-click.

REAL Studio

BBEdit Language Modules

Adding new language support to BBEdit and TextWrangler

by José R.C. Cruz

Introduction

Today we explore another way of extending BBEdit, the stalwart text editor of the MacOS X platform. This time, we will learn how to provide support to the editor for other languages.

To start, we take a close look at BBEdit's language feature. Then we learn two ways to build a language module. Next, we study the data structures that we need to process a target language. And we end by writing two language modules for that other OS X stalwart, AppleScript.

Readers are expected to know their way around Xcode and BBEdit. The modules featured here also work with BBEdit's free sibling, TextWranger, with some small variations. The Xcode projects for the modules are available from the following MacTech ftp site at ftp://ftp.mactech.com.

The Language Features

One notable feature of BBEdit is its ability to recognize and handle a wide range of computer languages. For instance, when it loads a text file written in a specific language, it marks the reserved keywords in the correct color.

Next, BBEdit can scan the text for valid blocks. A block can be a function, a property, or even a global. Once its scan is over, BBEdit displays the name of each block on a pop-up menu (Figure 1). Users can then jump to the desired block by selecting its name from the menu.

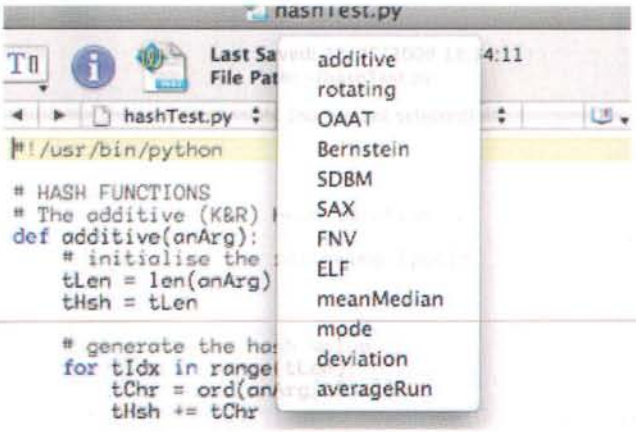


Figure 1. Displaying text blocks.

Another aspect of BBEdit's language feature is in its Search menu. For instance, choosing Go to Function Start places the insertion cursor before the block's *name* (Figure 2). Conversely, choosing Go To Function End places the cursor before the block's *closing token*. But choosing Go to Next Function selects the *name* of the next block. And choosing Go to Previous Function selects the name of the previous block.

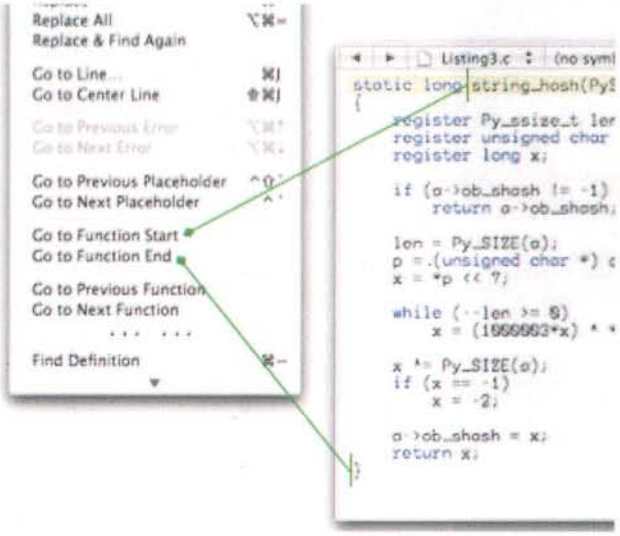


Figure 2. Navigating text blocks.

Finally, BBEdit has the ability to query a selected keyword from the right reference source. Suppose you selected the word "register" as shown in Figure 3. To start a query, bring up the contextual menu and choose the item Find in Reference. Since the target language is ANSI-C, BBEdit sends the query to Apple's Developer Connection website. If the language were Python, BBEdit will load Python's HTML documents, whose location is set by the environment variable `PYTHONDOCS`. Or if this is just a simple text file, BBEdit sends the query to the Dictionary application.

An Awesome Email Marketing Tool for MACTECH READERS

Benchmark Email is the standard in permission-based email marketing

- ✔ List Management
- ✔ Free Newsletter Archive
- ✔ Easy to Use Drag-n-Drop Email Editor
- ✔ Powerful Personalization
- ✔ Image Gallery
- ✔ Visual Graphs for Open & Clickthrough Tracking
- ✔ Spam & Spell Checkers
- ✔ Creative & Compelling HTML Templates
- ✔ Upload Your own Template
- ✔ Easy Video Integration

MACTECH users get a 10% Lifetime Discount with this promo code 123923

Most email marketing services charge more money for less product. Not us. Benchmark Email's sophisticated suite of email marketing features lets you grow your list, send campaigns, track your data and even take online polls for an affordable price.

Plans starting at only $9.99 per month

Sign up for a FREE 30 Day Trial Today!

www.BenchmarkEmail.com

benchmark email

800.430.4095

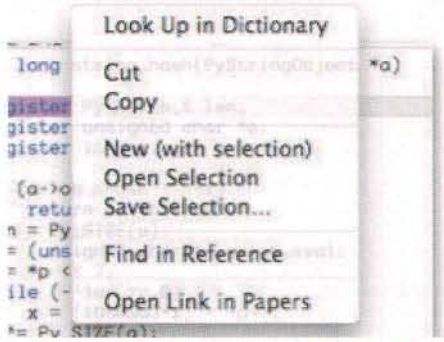


Figure 3. Cross-referencing a selected word.

Now BBEdit has built-in support for most common languages, which includes C, C++ and Objective-C. It even supports common scripting languages like Python, JavaScript, and bash. On the other hand, BBEdit lacks support for languages like AppleScript, Modula, and SPICE. To handle these niche languages, we must supply a custom language module.

The Language Module

The language module provides BBEdit with details about a target language. For instance, the module lists the reserved keywords that a given language uses. It sets the correct color schemes for each group of keywords. And it defines the block structures allowed by the language allows.

To use a language module, it must be inside the user directory `~/Library/ApplicationSupport/BBEdit/Language Modules/`. If this directory does not exist, you will have to create it using the Finder. Once you have installed the module, make sure to restart BBEdit in order to activate said module. To confirm if the module has activated, choose Preferences from the BBEdit menu. Then select the entry Languages to display the correct panel. On that panel, the module's target language should appear in the list labeled Installed languages (Figure 4).

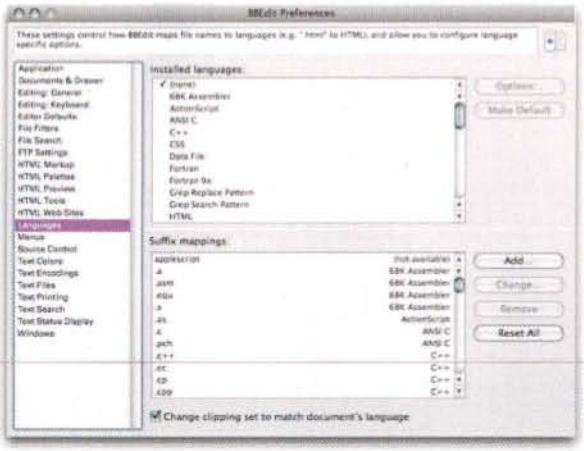


Figure 4. The Languages preferences panel.

The codeless module

There are two types of language modules. The first type, a *codeless module*, consists solely of one plist file. Inside this file are nine key/value pairs, most of which define an aspect of the target language. Table 1 lists the nine keys that the module uses. Keys that are not in this list are ignored by BBEdit.

Codeless modules are easy to assemble and test. All you need is a decent text editor, like BBEdit, and a good knowledge of the target language. On the other hand, these modules rely on BBEdit itself to correctly parse the language text. This means only versions 8.5 and newer of BBEdit can support codeless modules. For older versions of BBEdit, you may have to use a plug-in module.

The plug-in module

The *plug-in module* uses the same `CFPlugin` template as the tools plug-in. And it shares most of the key/value pairs that a codeless module uses. But unlike the latter, the plug-in module does the actual parsing the language text.

Figure 5 shows the structure of a typical plug-in module. The `MacOS` directory holds the module's executable binary. The `Resources` directory holds any support files that the module needs. The `Info.plist` file set the module's behaviour. It holds the key/value pairs that describe the target language. It also holds key/value pairs that describe the module itself. Table 2 lists some of the keys unique to the plug-in module.

The plug-in module can do certain tasks not possible with the codeless module. For example, it can color each keyword based on context. It can locate blocks nested in other blocks more accurately. And it can handle more than one target language. On the other hand, writing a plug-in module takes more time and resources than its codeless kin. Plus, a poorly written module may cause BBEdit to either freeze or crash unexpectedly.

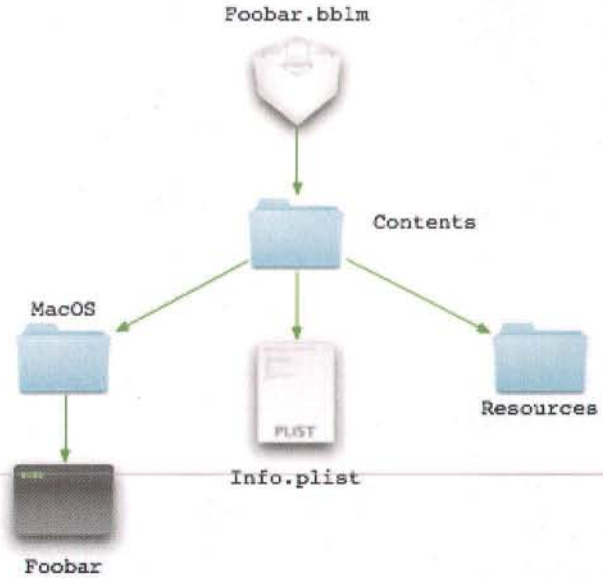


Figure 5. Structure of the plug-in module.

Table 1. Keys of the codeless module.		
Key	Type	Usage
`BBEditDocumentType`	string	Defines the type of language module. Always set to `CodelessLanguageModule`.
`BBLMColorsSyntax`	boolean	Controls syntax coloring. Set to `TRUE` to enable coloring.
`BBLMIsCaseSensitive`	boolean	Controls keyword parsing. Set to `TRUE` to enable case-sensitive parsing.
`BBLMKeywordList`	array	An array of core keywords for the target language. Must not include user-defined terms like variables and function names.
`BBLMLanguageCode`	string	Sets the target language's unique ID. Must be a four-byte string.
`BBLMLanguageDisplayName`	string	Sets the official name of the target language. This name appears in the Language panel of BBEdit's Preferences window.
`BBLMScansFunctions`	boolean	Enables function parsing.
`BBLMSuffixMap`	dictionary	Sets the file-name suffix used by the target language.
`LanguageFeatures`	dictionary	Sets the keywords and tokens that mark each language structure.

Table 2. Keys of the plug-in module.		
Key	Type	Description
`CFBundleIdentifier`	string	The plug-in module's unique ID, written as a reverse-domain URL.
`CFBundleSignature`	string	The plug-in module's unique type, always set to `BBLM`.
`CFBundleVersion`	string	The module's version string.
`CFResourcesFileMapped`	boolean	The access state of the module's plist resources. Always set to `TRUE`.
`com.barebones.bblminfo`	array	Start of the definitions of each target language.
`BBLMMainFunctionName`	string	The main entry function in the plug-in module. Must be unique for each target language.
`BBLMCanGuessLanguage`	boolean	Set to `TRUE` if the plug-in module can identify the target language based on context.

The Language SDK

To develop your own language module, you will need the latest SDK from Bare Bones Software. As it happens, that SDK is the same one used to develop a BBEdit tool plug-in. Not only does the SDK has the files needed by your language module, it also provides a template for a codeless module.

You can get the latest copy of the SDK from this URL.

http://www.barebones.com/support/develop/plugin_sdk.html

The SDK contents

The BBEdit SDK divides itself into four directories. The directory Codeless Examples holds three examples of codeless modules. It also holds the template file that forms the basis of those modules. The `Documentation` directory has the files that describe how to build the modules. The file of interest here is the one named "Writing BBEdit Language Modules" — it is available in both PDF and Microsoft Word format.

In the `Examples` directory are the project directories for three plug-in modules. You will need Xcode to these projects and study their settings. Finally, in the `Interface` directory are the header files that you will need to write your own module. These

files are separated into two sub-directories: `Tools` and `Languages`. The ones you should use are inside the `Languages` sub-directory.

The SDK header files

Now all plug-in modules use the header file `BBLMInterface.h`. This file defines the flags and constants that a plug-in can use. It lists the messages that same plug-in can expect from BBEdit. The file also defines the utility routines that can aid the plug-in in parsing the language text. And it sets the data structures wherein a plug-in can store its parsed data.

Some plug-in modules can also include the header file `BBLMTextIterator.h`, which defines the eponymous iterator class (Figure 7). This class supplies the means to parse and extract specific substrings from the given text. Plus, it can handle multi-byte text encodings. On the other hand, the class remains undocumented. To learn how to use this iterator class, consult the sample projects `PythonMachO.xcodeproj` and `TeXMachO.xcodeproj`.

Finally, plug-in modules can use the header file `BBXTInterface.h`. In this file are the constants, structures, and functions available to all BBEdit plug-ins. In this article, however, we will not use any of the services from `BBXTInterface.h` in our plug-in module.

Building The Codeless Module

Our first language module will use AppleScript as its target language. AppleScript, as all you may know, is the native scripting language of the MacOS platform. It first appeared in the mid-1990s as a feature of System 7 Pro, Apple's first commercial OS release. Unlike most script language at the time, AppleScript has an object-oriented syntax. It can control a target application using a structured messaging system called AppleEvents. It can even gain new features via plug-ins called *scripting additions*.

Now you can write the codeless module using any text editor, even BBEdit. You can also use the Property List Editor, part of Xcode's suite of tools, to edit your codeless module. If you decided to use BBEdit, keep in mind that BBEdit will not let you edit and test the module at the same time. A better approach is to write the module with one editor, such as TextWrangler, and then test the module on BBEdit.

Defining the language support

Start by copying the template file `CodelessLanguageModuleTemplate.plist`; then rename the copy as `AppleScript.plist`. Open `AppleScript.plist` in your text editor and search for the key `BBLMLanguageCode`. Set its value to "`ToyS`", the creator type for Apple's Script Editor. You can also use your

Be a hero.

CRASH**PLAN**PRO™
Continuous Backup for Business

Protect everyone, everywhere.

Only CrashPlan protects your workforce everywhere with real-time onsite and cloud backup.

Peace, justice and backup for all.

crashplanpro.com/hero

own four-character ID, but make sure to keep the ID unique to the target language.

Next, locate the key `BBLMLanguageDisplayName` and set its value to "`AppleScript`". At this point your module's first set of keys will have the settings shown in Listing 1. Notice we left four of the keys at their default values.

Listing 1. The principal module keys.

AppleScript.plist

```
<key>BBEditDocumentType</key>
<string>CodelessLanguageModule</string>
<key>BBLMColorsSyntax</key>
<true/>
<key>BBLMIsCaseSensitive</key>
<true/>
<key>BBLMLanguageCode</key>
<string>ToyS</string>
<key>BBLMLanguageDisplayName</key>
<string>AppleScript</string>
<key>BBLMScansFunctions</key>
<true/>
```

Now locate the key `BBLMSuffixMap`. Change its value as shown in Listing 2. This key hold an array of possible suffices for the source file. In the case of AppleScript, we assume the files to use `.applescript`. Other possible suffices include `.as`, `.scpt`, or `.applscrpt`. Again, be careful not to use the same suffices for two or more languages. Otherwise, BBEdit may use the wrong module to parse the source text.

Listing 2. Setting the file suffices.

AppleScript.plist

```
<key>BBLMSuffixMap</key>
<array>
    <dict>
        <key>BBLMLanguageSuffix</key>
        <string>.applescript</string>
    </dict>
</array>
```

Search for the key `BBLMKeywordList`, and enter the values shown in Listing 3. This is where we define the reserved words of the target language. The key holds an array of strings, with each element being a reserved word. All the reserved words are taken from the official Apple document "AppleScript Language Guide." The array shows only the reserved core words—it does not include words defined by a scriptable application or by a scripting addition.

Listing 3. Defining the reserved words (partial list).

AppleScript.plist

```
<key>BBLMKeywordList</key>
<array>
    <string>tell</string>
    <string>on</string>
    <string>to</string>
    <string>end</string>
    <string>return</string>
    <string>considering</string>
    <string>ignoring</string>
    <string>timeout</string>
    <string>transaction</string>

    <string>property</string>
    <string>global</string>
    <string>local</string>
    <!--
        for a complete list, see the sample project ...
    -->
</array>
```

Finally, locate the key Language Features and enter the key/value pairs listed by Listing 4. This is where we define much of AppleScript's syntax structure. For instance, block comments in AppleScript start with a `'(*'` token and ends with a `'*)'`. But inline comments start only with a `'--'` token. Also, AppleScript variables and values are written in alphanumeric characters. Plus, variables can use a `'_'` token (`0x5f`) to separate parts of their names. Functions and procedures, called *handlers* in AppleScript, begin with the reserved word prefixes `'on'` or `'to'`. Statement blocks start with a `'tell'` prefix and end with an `'end tell'` prefix.

Listing 4. Defining the language features (partial list).

AppleScript.plist

```
<key>Language Features</key>
<dict>
    <key>Prefix for Functions</key>
    <string>on</string>
    <key>Prefix for Procedures</key>
    <string>to</string>
    <key>Close Block Comments</key>
    <string>*)</string>
    <key>Close Parameter Lists</key>
    <string>)</string>
    <key>Close Statement Blocks</key>
    <string>end tell</string>
    <key>Close Strings 1</key>
    <string>"</string>
    <key>Close Strings 2</key>
    <string>»</string>
    <key>Identifier and Keyword Characters</key>
    <string>0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
        _abcdefghijklmnopqrstuvwxyz</string>
    <key>Open Block Comments</key>
    <string>(*</string>
    <key>Open Line Comments</key>
    <string>--</string>
    <key>Open Parameter Lists</key>
    <string>(</string>
    <key>Open Statement Blocks</key>
    <string>tell</string>
    <key>Open Strings 1</key>
    <string>"</string>
    <key>Open Strings 2</key>
    <string>«data</string>
    <!--
        for a complete list, see the sample project ...
    -->
</dict>
```

Now *readable strings* in AppleScript are enclosed in double quotes (`0x22`), just like in other modern languages. If the string uses special symbols, each symbol will be preceded by the escape token `'\'`. Then there are the *raw data strings*. These start with a `'«data'` token and ends with a `'»'` token. In between these tokens are the data bytes rendered as a hexadecimal string. Finally, it is possible for a string to contain a newline character (`0x0a`). In this case, the closing quote will appear on a separate line.

Installation and testing

To test the codeless module, first copy or move the plist file to the following directory.

`~/Library/Application Support/BBEdit/Language Support/`

Quit BBEdit if it is still running; then relaunch it to enable the module. Now choose **Preferences** from the BBEdit menu. From the list of preferences panels, select the entry **Languages**. You should see an entry for AppleScript on the listbox **Installed Languages**. And in the listbox **Suffix Mappings**, you should find the suffix **.applescript** as one of the entries (Figure 6).

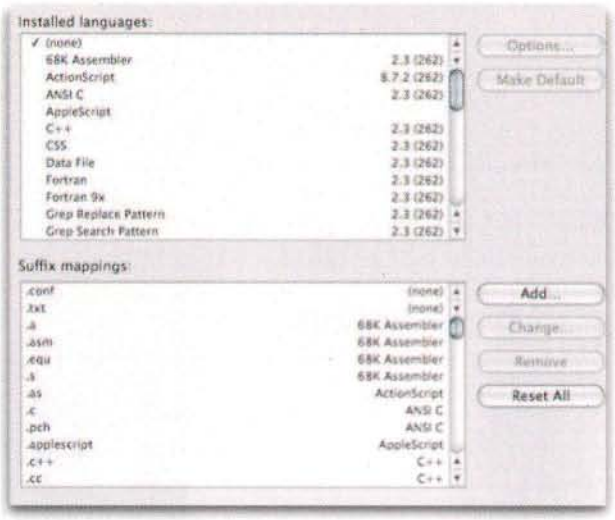


Figure 6. Looking for the AppleScript codeless module.

Now create a new document window by choosing New Text Document from the File menu. Then enter the text shown in Listing 5. Notice the entered text is still uncolored. This is because you have yet to save the document and assign the correct suffix.

Listing 5. The test script.

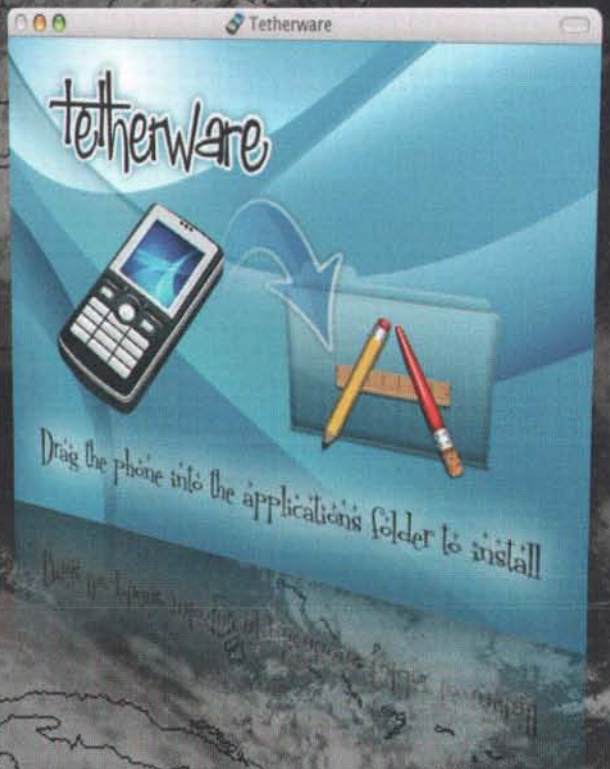
foobar.applescript

```
(*
    This is a block comment
*)
property gRoot : "OS X:Applications:Chess.app:Contents:"
set tTest to path to documents folder
set tTest to gRoot as string

(*
    This is another block comment
*)
tell application "Finder"
    set tFoo to "--foo"
    set tBar to "foo \"bar\" foo"
    kind of alias gRoot
end tell

— This is an inline comment
to listFoo from aSrc
    local tLst

    tell application "System Events"
        set tLst to every item of alias aSrc
    end tell — application "System Events"
end listFoo — from aSrc

(* This is also a block comment *)
on listBar(aSrc)
    local tLst

    tell application "Finder"
        set tLst to every file in alias aSrc as list
    end tell — application "Find"
end listBar — (aSrc)
```

Go to the File menu and choose the menu item Save As. Set the file name to `foobar.applescript`, but leave the file location to your home directory. Click the Save button to create the script file. Once BBEdit writes the script file, it automatically renders the text as shown in Figure 7. Furthermore, it marks the start of each function block with a triangle icon and the end of each block with an invert-L icon. Clicking the triangle icon collapses the block; clicking it again expands the block. Finally,

Does your DMG look damaged?

DMG built WITHOUT FileStorm in OS X 10.6
displayed on 10.5 or earlier



DMG built WITH FileStorm in OS X 10.6
displayed on 10.5 or earlier

FileStorm from MindVision Software.

For when your image is hurting your image.

Purchase and download now
www.mindvision.com/mactech

click the function pop-up list on the window. You will see the names `listFoo` and `listBar` appear on that list.

```
    this is a block comment
*)
property gRoot : "OS X:Application
set tTest to path to documents fol
set tTest to gRoot as string
(*
    This is another block comment
*)
tell application "Finder"
    set tFoo to "---foo"
    set tBar to "foo \"bar\" foo"
    kind of alias gRoot
end tell
-- This is an inline comment
to listFoo from aSrc
    local tLst
    tell application "System Event
        set tLst to every item of
    end tell -- application "Syste
end listFoo -- from aSrc
```

Figure 7. The rendered AppleScript script.

But note how BBEdit marks line 7 with a triangle icon. This is incorrect, of course, for the line does not start a function block. What happened here is that the keyword `'to'` has two roles; a prefix of a function block (line 14) or an assignment (lines 6 and 7). Unfortunately, the codeless module could not differentiate between these two roles. One solution is to treat `'to'` only as an assignment keyword. A better way is to write the module as a plug-in module.

Preparing The Plug-in Module

Now let us build our AppleScript module as a plug-in module. Start up Xcode and choose New Project from its File menu. Scroll down the list of project templates and choose the template `CFPlugin`. Click the Next button and set the project name to `AppleScript`. Leave the project location set to your home directory. Then click the Finish button to create the plug-in project.

Go to the Groups & Files pane and select the entry `InfoPlist.strings`. Set the key `NSHumanReadableCopyright` to the desired copyright text. Then choose Edit Active Target from the Project menu. Scroll down until you find the `Packaging` group. Change the value of the entry Wrapper Extension to `bblm`. Close the target settings window and save your changes.

Next, switch to the Finder and go to the `AppleScript` project directory. Create an empty directory and name it `Headers`. To this directory, copy the following header files from the BBEdit SDK.

```
BBLMInterface.h, BBXTInterface.h,
BBXTImplementationMacros.h
BBXTImplementationStructsAndEnums.h
```

Switch back to Xcode and choose Add to Project from the Xcode's Project menu. Use the Open File dialog to select the files inside the `Headers` directory. Click the Add button to accept these files. When you do, make sure you have AppleScript as the file's target.

Finally, choose New File from the File menu. Select `C++ File` from the list of file templates. Click the Next button and set the file name to `AppleScript.cp`. Make sure to allow Xcode to create a header file. Then click the Finish button to create the project source file. When done, your project window should appear as shown in Figure 8.

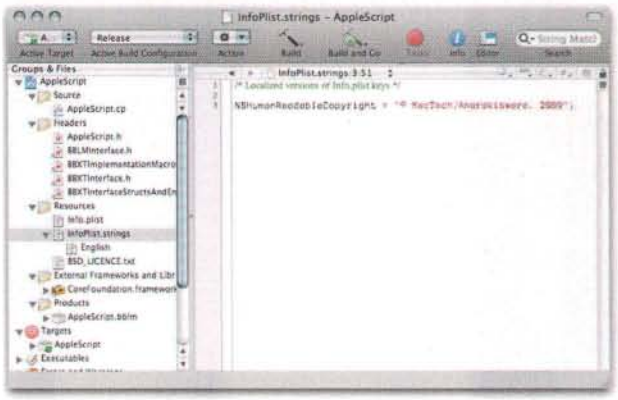


Figure 8. The Xcode project (AppleScript.xcodeproj).

The Info.plist file

Select the entry `Info.plist` from the Groups & Files pane. Locate the key `CFBundleIdentifier`; set its value to `com.mactech.anarakisware.demo.applescript`. If you prefer, you can supply your own unique bundle ID. Next, set the key `CFBundleSignature` to `BBLM`. Then set the key `CFBundleVersion` to `1.0.0d1` or to your own version string. Leave the rest of the `CFBundle` keys to their default values.

Scroll down to the end of the `Info.plist` file, and enter the key/value pairs shown in Listing 6. This group defines the basic aspects of the plug-in, just like Listings 1 and 2. It also sets `AppleScriptMain` as the plug-in's main entry function, which we will cover next.

Listing 6. Describing the plug-in module.

Info.plist

```
<key>com.barebones.bblminfo</key>
<array>
    <dict>
        <!— Plug-in primer —>
        <key>BBLMLanguageCode</key>
        <string>ToyS</string>
        <key>BBLMLanguageDisplayName</key>
        <string>Applescript</string>
        <key>BBLMMainFunctionName</key>
        <string>AppleScriptMain</string>
        <key>BBLMSuffixMap</key>
        <array>
```

```
            <dict>

<key>BBLMLanguageSuffix</key>

<string>.applescript</string>
            </dict>
        </array>

        <!— Plug-in attributes —>
        <!— see Listing 7 —>

        <!— Reserved words —>
        <!— see Listing 8 —>
    </dict>
</array>
```

Next, enter the key/value pairs shown in Listing 7. This second group defines the plug-in's behaviour. Our plug-in is expected to scan for function blocks and to color language keywords. It does not have to do case-sensitive tasks or guess the target language in use. But it does assume the path of the target file to be in POSIX form.

Listing 7. Defining the plug-in's behavior.

Info.plist

```
<key>com.barebones.bblminfo</key>
<array>
    <dict>
        <!— Plug-in primer —>
        <!— see Listing 6 —>

        <!— Plug-in attributes —>
        <key>BBLMScansFunctions</key>
        <true/>
        <key>BBLMCanGuessLanguage</key>
        <false/>
        <key>BBLMColorsSyntax</key>
        <true/>

<key>BBLMDroppedFilePathStyle</key>
        <string>POSIX</string>
        <key>BBLMIsCaseSensitive</key>
        <true/>

<key>BBLMUseHTMLFileSearchRules</key>
        <false/>

        <!— Reserved words —>
        <!— see Listing 8 —>
    </dict>
</array>
```

Now enter the key/value pairs in Listing 8. This group defines the core keywords that make up the target language. It serves the same role as Listing 3 of the codeless module.

Listing 8. Defining the language keywords (partial list).

Info.plist

```
<key>com.barebones.bblminfo</key>
<array>
    <dict>
```

Listing 9. The main entry function.

AppleScriptMain()

```
extern "C"
{

    OSErr AppleScriptMain(BBLMParamBlock &aArg
                          , const BBLMCallbackBlock &aBLM
                          , const BBXTCallbackBlock &aBXT)
    {
        OSErr result;

          // validate the parameter block
        if ((aArg.fSignature != kBBLMParamBlockSignature) ||
            (aArg.fVersion < kBBLMParamBlockVersion))
        {
            return paramErr;
        }

          // identify the plug-in message
        switch (aArg.fMessage)
        {
            case kBBLMInitMessage:
                    // — the plug-in module is loading
            case kBBLMDisposeMessage:
                    // — the plug-in module is unloading
                result = noErr;
                break;

            case kBBLMScanForFunctionsMessage:
                    // — a list of function names is needed
                ASFunctionScan(aArg, aBLM);
                break;

            case kBBLMCalculateRunsMessage:
                    // — a list of syntax runs is needed
                ASCalculateRuns(aArg, aBLM);
                break;

            case kBBLMAdjustRangeMessage:
                    // — the indices of the first and last language run has changed
            case kBBLMAdjustEndMessage:
                    // — the offset to the last txt character has changed
            case kBBLMMapRunKindToColorCodeMessage:
                    // — map a user-defined run code to a run-color
            case kBBLMMapColorCodeToColorMessage:
                    // — map a user-defined run color to an actual color
            case kBBLMEscapeStringMessage:
                    // — handle an escape string character
            case kBBLMSetCategoriesMessage:
                    // — configure character categories
            case kBBLMMatchKeywordMessage:
                     // — look for a keyword match
            case kBBLMGuessLanguageMessage:
                      // — identify the target language
                result = userCanceledErr;
                break;

            default:
                 // plug-in:message:unknown/unsupported
            {
                result = paramErr;
                break;
            }
        }

        return result;
    } // OSErr AppleScriptMain

} // extern "C"
```

Listing 10. Looking for function names.

ASFunctionScan()

```
OSErr ASFunctionScan(BBLMParamBlock &aArg, const BBLMCallbackBlock *aBLM)
{
    CFRange tRng;
    CFIndex tLen, tPos;
    CFStringRef tSrc, tLin, tNom;
    CFStringTokenizerRef tSen, tWrd;
    CFStringTokenizerTokenType tChk;
     OSErr tErr;

     // read the target text
    tSrc = CFStringCreateWithCharacters(kCFAllocatorDefault
                              , (UniChar*) aArg.fText
                              , aArg.fTextLength);
    // parameter check
    tLen = CFStringGetLength(tSrc);
    if (tLen > 0)
    {
          // create a line tokenizer
        tRng = CFRangeMake(0, tLen);
        tSen = CFStringTokenizerCreate(kCFAllocatorDefault
                                    , tSrc, tRng
                                    , kCFStringTokenizerUnitParagraph
                                    , CFLocaleCopyCurrent());
        if (tSen != NULL)
        {
            CFRange tFnd;

                // parse the source text
            tPos = 0;
            while (CFStringTokenizerAdvanceToNextToken(tSen)
                    != kCFStringTokenizerTokenNone)
            {
                    // extract a source line
                tRng = CFStringTokenizerGetCurrentTokenRange(t$en);
                tLin = CFStringCreateWithSubstring(kCFAllocatorDefault
                                                , tSrc
                                                , tRng);

                    // scan the line for a function prefix
                tFnd = CFStringFind(tLin, CFSTR("to"), 0);
                if (tFnd.location == kCFNotFound)
                    tFnd = CFStringFind(tLin, CFSTR("on"), 0);

                    // was the check successful?
                if (tFnd.location == 0)
                {
                        // create a word tokenizer
                    tRng = CFRangeMake(0, CFStringGetLength(tLin));
                    tWrd = CFStringTokenizerCreate(kCFAllocatorDefault
                                                , tLin, tRng
                                                , kCFStringTokenizerUnitWord
                                                , CFLocaleCopyCurrent());
                    if (tWrd != NULL)
                    {
                            // locate the function name
                        tChk = CFStringTokenizerAdvanceToNextToken(tWrd);
                        tChk = CFStringTokenizerAdvanceToNextToken(tWrd);
                        tRng
                            = CFStringTokenizerGetCurrentTokenRange(tWrd);
                        tNom
                            = CFStringCreateWithSubstring(kCFAllocatorDefault
                                                , tLin
                                                , tRng);
                            // update the function list
                        tErr = ASFunctionList(aArg, aBLM, tNom
                                                , (tPos + 3));

                    } // (tWrd != NULL)
                } // (tFnd.location == 0)

                    // update the line position
                tPos += CFStringGetLength(tLin);
            } // while(...)
        } // (tSen != NULL)
    } // (tLen > 0)

    // return the function results
    return (tErr);
} // OSErr ASFunctionScan()
```

```
            <!— Plug-in primer —>
            <!— see Listing 6 —>

            <!— Plug-in attributes —>
            <!— see Listing 7 —>

            <!— Reserved words —>
            <key>BBLMKeywordList</key>
            <array>

<string>tell</string>
                        <string>on</string>
                        <string>to</string>

<string>end</string>

<string>return</string>

<string>considering</string>

<string>ignoring</string>

<string>timeout</string>

<string>transaction</string>


<string>property</string>

<string>global</string>

<string>local</string>

                        <!— for the complete
list, see the Xcode project —>
            </array>
        </dict>
    </array>
```

The main entry function

Select the entry `AppleScript.cp` from the Groups & Files pane. Then enter the code shown in Listing 9. The code defines the basic structure of the main entry function `AppleScriptMain`. The function takes three input arguments. The first argument is an instance of the `BBLMParamBlock` struct. This struct supplies the text data for the plug-in to process. It instructs the plug-in on how to process the data, and it tells the plug-in where to display the result. *See Listing 9 below.*

The second input argument is an instance of the struct `BBLMCallbackBlock`. This struct defines the utility methods that a language plug-in can use. As a rule, you do not access those methods directly from `BBLMCallbackBlock`. Use instead the various inline methods defined in the header file `BBLMInterface.h`. These inline

methods require the `BBLMCallbackBlock` as one of their inputs.

The third input argument is an instance of the `BBXTCallbackBlock` struct. As stated earlier, this struct supplies the inline methods that the plug-in can use. Information about these inline methods can be found in the file `BBXTInterface.h`.

Next, the main entry function uses an `OSErr` as its return value. BBEdit uses this value to find out how the plug-in handled its tasks. For instance, if the plug-in gets an older, incompatible version of `BBLMParamBlock`, it returns a `paramErr`. If it ignores a specific message, it returns a `userCanceledErr`. If it handles a message without problems, it returns a `noErr`. For a list of other possible errors, consult the Carbon header file `MacErrors.h`.

Finally, the main entry function checks the `fMessage` field of `BBLMParamBlock`. This field specifies the action that BBEdit wants the plug-in to do. The action can be a simple one like a start-up, or it can be something more complex like a function parse. The file `BBLMInterface.h` describes all possible actions that a plug-in module can expect. The plug-in can choose to handle all these actions, or just the base subset. The AppleScript module, for instance, will handle only the actions `kBBLMScanForFunctionsMessage` and `kBBLMCalculateRunsMessage`.

Now compile the project by choosing **Build** from the Build menu. Xcode will then create the plug-in bundle and name it `AppleScript.bblm`. Switch to the Finder, locate and copy the bundle to BBEdit's assigned directory for its language modules. Remove the codeless module `AppleScript.plist` if one is in the directory. Restart BBEdit and check the **Languages** panel of its **Preferences** window. You should find AppleScript listed on the **Installed languages** widget.

Listing 11. Listing the function names.

ASFunctionList()

```
OSErr ASFunctionList(BBLMParamBlock &aArg, ¬
const BBLMCallbackBlock *aBLM
                    , CFStringRef aNom, CFIndex aPos)
{
    BBLMProcInfo tInf;
    CFIndex tLen;
    UInt32 tIdx = 0;
    OSErr tErr = userCanceledErr;

      // initialise the following locals
    tLen = CFStringGetLength(aNom);

      // reset the proc-info block
    memset(&tInf, 0, sizeof(tInf));

      // set the following block fields
    tInf.fFunctionStart = aPos;
    tInf.fFunctionEnd = tInf.fFunctionStart + tLen;
    tInf.fSelStart = aPos;
    tInf.fSelEnd = tInf.fSelStart + tLen;
    tInf.fFirstChar = aPos;
    tInf.fIndentLevel = 0;
    tInf.fKind = kBBLMFunctionMark;
    tInf.fFlags = 0;
    tInf.fNameStart = 0;
    tInf.fNameLength = tLen;

      // update the list buffer
    tErr = bblmAddCFStringTokenToBuffer(aBLM
                        , aArg.fFcnParams.fTokenBuffer
                        , aNom
                        , &tInf.fNameStart);
    if ( tErr == noErr )
        tErr = bblmAddFunctionToList(aBLM
                                , aArg.fFcnParams.fFcnList
                                , tInf
                                , &tIdx);
      // return the update result
    return (tErr);
}
```

Handling The Plug-in Messages

Since we have a working plug-in module, we are now ready to handle the various messages that BBEdit sends to our plug-in. To keep things simple, we will focus only on two messages: `kBBLMScanForFunctionsMessage` and `kBBLMCalculateRunsMessage`. BBEdit sends the first message when it wants a list of function names from the plug-in. It sends the second message when it wants the plug-in to identify specific regions of the target text.

Scanning for functions

To handle the message `kBBLMScanForFunctionsMessage`, the plug-in runs the routine `ASFunctionScan()` (Listing 10). This routine takes two input parameters, which are instances of `BBLMParamBlock` and of `BBLMCallbackBlock`. Its output is an `OSErr` constant.

The routine starts by storing the target text into an instance of CFString. Then it creates an instance of CFStringTokenizer, passing the CFString object as input. It also configures the tokenizer to divide the target text into separate lines of code. Next, `ASFunctionScan()` parses each line of target text. If a line starts with either a `'to'` or an `'on'`, the routine extracts the line and uses it to create a *second* CFStringTokenizer. Then it sets that tokenizer to divide the line into its constituent words. The routine extracts the *second* word after the `'to/on'` prefix and passes the results to the routine `ASFunctionList()`.

The `ASFunctionList()` routine (Listing 11) handles the update of BBEdit's function pop-up (see Figure 2). It takes four parameters: the function name, its position on the text, and the same two structs that `ASFunctionScan()` gets. Like `ASFunctionScan()`, the routine returns its result as an `OSErr`.

Listing 12. Handling the syntax run message.

ASCalculateRuns()

```
OSErr ASCalculateRuns(BBLMParamBlock &aArg
                        , const BBLMCallbackBlock *aBLM)
{
    CFRange tRng;
    CFIndex tLen, tPos;
    CFStringRef tSrc, tLin;
    CFStringTokenizerRef tSen;

    OSErr tErr = userCanceledErr;

      // read the target text
    tSrc = CFStringCreateWithCharacters(kCFAllocatorDefault
                        , (UniChar*) aArg.fText
                        , aArg.fTextLength);

      // parametre check
    tLen = CFStringGetLength(tSrc);
    if (tLen > 0)
    {
          // create a line tokenizer
        tRng = CFRangeMake(0, tLen);
        tSen = CFStringTokenizerCreate(kCFAllocatorDefault
                                    , tSrc, tRng
                                    , kCFStringTokenizerUnitParagraph
                                    , CFLocaleCopyCurrent());
        if (tSen != NULL)
        {
            CFRange tBgn, tEnd;
            Boolean tBbc = false;

            BBLMRunKind tTyp;

                // parse the source text
            tPos = 0;
            while (CFStringTokenizerAdvanceToNextToken(tSen)
                    != kCFStringTokenizerTokenNone)
            {
                    // extract a source line
                tRng = CFStringTokenizerGetCurrentTokenRange(tSen);
                tLin = CFStringCreateWithSubstring(kCFAllocatorDefault
                                                , tSrc
                                                , tRng);

                    // checking for a block comments
                if (tBbc)
                {
                    // check for the end of block comments
                    tEnd = ASScanBlockComments(tLin, true);
                    if (tEnd.location != kCFNotFound)
                    {
                            // measure the run
                        tTyp = kBBLMRunIsBlockComment;
                        tBgn.length = CFStringGetLength(tLin);

                            // mark the run
                        tErr = ASMarkRun(aArg, aBLM, tTyp, tPos, tBgn);

                            // end the run
                        tBbc = false;
                    }
                }
                else
                {
                        // check for the start of block comments
                    tBgn = ASScanBlockComments(tLin, false);
                    tBbc = (tBgn.location != kCFNotFound);
                    if (tBbc)
                    {
                            // check for the end of block comments
                        tEnd = ASScanBlockComments(tLin, true);
                        if (tEnd.location != kCFNotFound)
                        {
                                // measure the run
                            tTyp = kBBLMRunIsLineComment;
```

Listing 12 continues

The routine starts by preparing an empty instance of the `BBLMProcInfo` struct. It updates the fields in that struct with data showing the position of the function name. Then it calls the inline function `bblmAddCFStringTokenToBuffer()`, passing the function name and position as input. Finally, it calls the inline function `bblmAddFunctionToList()`, passing the `BBLMProcInfo` instance as input. Notice that both inline functions return an `OSErr` as their result.

Scanning for syntax runs

Now to handle the message `kBBLMCalculateRunsMessage`, the plug-in runs the routine `ASCalculateRuns()` (Listing 12). This routine also takes the instances of `BBLMParamBlock` and `BBLMCallbackBlock` as input. And it returns an `OSErr` as its result.

The `ASCalculateRuns()` method follows a similar code structure as `ASFunctionScan()`. It uses an instance of `CFStringTokenizer` to divide the source text into distinct lines of code. Then it tests if each line forms part of a block comment. If the test fails, the method tests if the line form an inline string or comment. When the line passes either tests, `ASCalculateRuns()` calls the method `ASMarkRun()`. And `ASCalculateRuns()` passes the type of syntax run, its location and length.

The `ASCalculateRuns()` method uses the `ASScanBlockComments()` function (Listing 13) to identify a block comment. This function takes two input parameters: a source line and a Boolean flag. This flag tells the function if it should look for the start or end of a block comment. If the flag is set to `FALSE`, the function checks if the source line starts with a `'(*'` token. If the flag is set to `TRUE`, the function checks if the source line ends with a `'*)'` token. Then it returns its search result as a `CFRange`.

To look for inline strings or comments, `ASCalculateRuns()` uses the `ASScanInline()` function (Listing 14). This function also takes two input arguments: a source line and a range. But it returns its search results as a `BBLMRunKind`. Note the input argument `aRng` is passed by reference. This allows `ASScanInline()` to update the argument with its its search results.

`ASScanInline()` starts by looking for either a `'–'` or a `'"'` token in the source line.

A '–' token means the line *has* or *is* an inline comment. A '"' token means it has an inline string. If the line contains either one, the function decides which token appeared first. Next, `ASScanInline()` scans the extent of the inline structure. In the case of an inline comment, the function simply looks for the end-of-line token. In the case of an inline string, the function looks for a closing '"' token. But it ignores any '"' token that comes after a '\'.

Listing 15 shows the utility function `ASMarkRuns()`. The function takes five input arguments, the first two being the instances of `BBLMParamBlock` and `BBLMCallbackBlock`. The other three arguments specify the type of run, its starting position, and its length. First, the function identifies what run it should process. Next, it calculates the actual position of the run in the source text and the number of characters contained. Then it calls the BBEdit function `bblmAddRun()`, which marks the run with the right text color.

For reasons of length, `ASMarkRuns()` only handles three types of runs. For a list of other run types, check the enum `BBLMRunKind` in the header file `BBLMInterface.h`. Supporting these runs will be left as an exercise to the reader.

Installation and testing

Recompile the plug-in module by choosing Build from the Build menu. Then copy the `AppleScript.bblm` bundle into the directory `~/Library/ApplicationSupport/BBEdit/Language Modules`. Make sure you remove the older bundle before copying the new one. If BBEdit is already running, choose Quit from its application menu. Then restart BBEdit and bring up its preferences window. Check the Languages preferences panel and see if AppleScript is still listed in the Installed Languages listbox.

Now go to the File menu and click the sub-menu Open Recent sub-menu. Choose the entry `foobar.applescript` to bring up the text file. Notice how BBEdit marks the block comments and some of the inline comments to the right colour (Figure 11). But note that it failed to do the same for the other inline comments and all the inline strings. This is a side effect of how BBEdit handles its text data.

To improve display and typing performance, BBEdit inserts *gaps* within each block of text. These gaps contain *non-readable* characters and have arbitrary position and lengths. But both `CFString` and its associate routines are unable

Listing 12 continued.

```
                        tBgn.length = CFStringGetLength(tLin);

                                    // mark the run
                        tErr = ASMarkRun(aArg, aBLM, tTyp
                                                , tPos, tBgn);

                                    // end the run
                        tBbc = false;
                    }
                    else
                    {
                                    // save the run
                        tBgn.location = tPos;
                        tBgn.length = CFStringGetLength(tLin);
                    }
                }
                else
                {
                                // check for an inline comment or string
                    tTyp = ASScanInline(tLin, &tBgn);

                                // mark the run
                    tErr = ASMarkRun(aArg, aBLM, tTyp, tPos, tBgn);
                } // if (tBbc)
            } // if (tBbc)

        // update the line position
            tPos += CFStringGetLength(tLin);
        } // while(...)
    } // (tSen != NULL)
} // (tLen > 0)

    return (tErr);
}
```

Listing 13. Scanning for block comments.

ASScanBlockComments()

```
CFRange ASScanBlockComments(CFStringRef aSrc, Boolean aEnd)
{
    CFRange tRng;
    Boolean tFlg;

      // set the search position
    tRng = CFRangeMake(0, CFStringGetLength(aSrc));

      // is this the start or the end of the search?
    if (aEnd)
          // run a backward search
        tFlg = CFStringFindWithOptions(aSrc
                        , CFSTR("*)")
                        , tRng
                        , kCFCompareBackwards || kCFCompareAnchored
                        , &tRng);
    else
          // run a forward search
        tFlg = CFStringFindWithOptions(aSrc
                        , CFSTR("(*")
                        , tRng
                        , kCFCompareAnchored
                        , &tRng);

      // return the parse results
    if (tFlg)
        return (tRng);
    else
        return (CFRangeMake(kCFNotFound, 0));
```

Listing 14. Scanning for inline comments or strings.

ASScanInline()

```
BBLMRunKind ASScanInline(CFStringRef aLin, CFRange *aRng)
{
    CFRange tBic, tBst;
    CFIndex tLen;
    UniChar tChr;

    BBLMRunKind tTyp = kBBLMRunIsCode;

      // look for a starting token
    tBic = CFStringFind(aLin, CFSTR("--"), 0);
    tBst = CFStringFind(aLin, CFSTR("\""), 0);

      // validate the token positions
    if (tBic.location == kCFNotFound)
    {
        if (tBst.location != kCFNotFound)
        {
                  // locate the end of the string
            CFIndex tPos;

            tPos = tBst.location;
            tLen = CFStringGetLength(aLin) - 1;
            while (tPos < tLen)
            {
                tChr = CFStringGetCharacterAtIndex(aLin, tPos);
                if ((tPos > tBst.location)
                    && (tChr == 0x22))
                {
                            // read the preceding character
                    tChr = CFStringGetCharacterAtIndex(aLin, tPos - 1);
                    if (tChr != 0x5c)
                        break;
                }
                tPos++;
            }
             // measure the length of the inline string
```

Listing 14 continues

to handle these gaps. Plus, BBEdit leaves these gaps in place when it supplies its text data to the plug-in module. So, the plug-in must know how to detect and remove these gaps before it processes a syntax run.

Thankfully, the `BBLMParamBlock` structure holds the gap information in two fields. The first field `fTextGapLocation` is the *position of the gap* from the first character of the target text. The second field `fTextGapLength` is the *number of characters* within that gap. Use these fields to strip off any gaps prior to handling the text. This will be left as an exercise to the readers.

Another way to handle the gaps is to use `BBLMTextIterator` instead of `CFStringTokenizerRef` to parse the target text. `BBLMTextIterator` has a distinct advantage of being gap-aware. Again, consult the BBEdit SDK for examples on how to use this custom iterator.

Concluding Remarks

Throughout this article, we examined how BBEdit handles a document written in a specific computer language. We saw how it colors each language keyword or block, and how it lists any function names present. We even learned how it aids users move from one function block to another.

Yet, we find that BBEdit does not support some niche languages such as AppleScript. So to address this lack, we learned how to write two types of language modules. One module uses BBEdit's ability to parse the target text; the other handles the actual parsing. We also learned how to install and test either module on BBEdit.

And that ends our coverage of BBEdit's language modules. You can find examples of other language modules from the Bare Bones web site at the following URL. http://www.barebones.com/support/bbedit/plugin_library.html

My thanks go to Patrick Woolsey of Bare Bones Software for his help in making a viable language plug-in. Special thanks goes to Seth Dillingham of MacroByte Resources for his insights on the gap issue and its effects on the language plug-in.

Until next time, take care.

Bibliography and References

Apple Developer Connections. *AppleScript Language Guide*. Cupertino, CA:Apple Computers Inc, 2003-2006. [Online]. Available:

Listing 14 continued

```
            tLen = tPos - tBst.location;
            aRng->location = tBst.location - 1;

            aRng->length = tLen;

                // set the return type
            tTyp = kBBLMRunIsDoubleString;
}
    }
    else
    {
        if ((tBst.location != kCFNotFound)
            && (tBst.location < tBic.location))
        {
                // locate the end of the string
            CFIndex tPos;

            tPos = tBst.location;
            tLen = CFStringGetLength(aLin) - 1;
            while (tPos < tLen)
            {
                tChr = CFStringGetCharacterAtIndex(aLin, tPos);
                if ((tPos > tBst.location)
                    && (tChr == 0x22))
                {
                        // read the preceding character
                    tChr = CFStringGetCharacterAtIndex(aLin, tPos - 1);
                    if (tChr != 0x5c)
                        break;
                }
                tPos++;
            }
                // measure the length of the inline string
            tLen = tPos - tBst.location;
            aRng->location = tBst.location;
            aRng->length = tLen + 1;

                // set the return type
            tTyp = kBBLMRunIsDoubleString;
        }
        else
        {
                // measure the length of the inline block
            tLen = CFStringGetLength(aLin) - tBic.location - 1;
            aRng->location = tBic.location;
            aRng->length = tLen;

                // set the return type
            tTyp = kBBLMRunIsLineComment;
        }
    }

    // return the scan results
    return (tTyp);
}
```

Listing 15. Marking a syntax run.

ASMarkRun()

```
OSErr ASMarkRun(BBLMParamBlock &aArg
                        , const BBLMCallbackBlock *aBLM
                        , BBLMRunKind aTyp
                        , CFIndex aPos, CFRange aRng)
{
    CFIndex tPos, tLen;
    bool    tChk;

    // submit the syntax run
    switch (aTyp)
    {
        case kBBLMRunIsLineComment:
        case kBBLMRunIsDoubleString:
            tPos = aPos + aRng.location;
            tLen = aRng.length;
            tChk = bblmAddRun(aBLM, aArg.fLanguage
                                , aTyp
                                , tPos, tLen
                                , false);
            break;

        case kBBLMRunIsBlockComment:
            tPos = aRng.location;
            tLen = aPos - tPos + aRng.length;
            tChk = bblmAddRun(aBLM, aArg.fLanguage
                                , aTyp
                                , tPos, tLen
                                , false);
            break;

        default:
            tChk = true;
            break;
    }

    // return the mark results
    return (noErr);
}
```


http://developer.apple.com/documentation/applescript/Conceptual/AppleScript-LangGuide/introduction/ASLR_intro.html

Apple Developer Connections. "Plug-ins", *Core Foundation: Process Management*. Cupertino, CA:Apple Computers Inc, 2003-2006. [Online]. Available: http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFPlugIns/CFPlugIns.html

Bare Bones Software. *Writing Language Modules for BBEdit and TextWrangler*. BedFord, MA:Bare Bones Software, 2006.

Bare Bones Software. "Codeless Language Modules," in *BBEdit User Manual*. BedFord, MA:Bare Bones Software, 2006, pp. 357-365.

About The Author

JC is a freelance engineering writer from North Vancouver, British Columbia. He writes for various publications, covering diverse topics such as AppleScript, REALbasic, Python, and Cocoa. He also spends quality time with his foster nephew. You can reach JC at anarakisware@gmail.com.

MacTech Spotlight: Boisy G. Pitre
Continues from page 80

created on my Mac, and used DriveWire as the media platform for testing (see http://www.apple.com/science/poster/)

Ever?

For fun, I've restored several vintage Coca-Cola machines, and reworked a crusty old Pac-Man arcade cabinet and put a PC running MAME inside, complete with working coin door and controls (it's the only thing that runs Windows here). Is that tech or what?

Where can we see a sample of your work?

With a name like mine, all you need to do is Google me, and you'll see link after link of things that I've worked on over the years.

The next way I'm going to impact IT/OS X/the Mac universe is:

Besides continuing to work on cutting edge software, I am interested in leveraging my experience on the Mac and iPhone platforms by writing technical articles for magazines (like MacTech). It's very rewarding to be able to use what you've learned and pass it on to others.

If you or someone you know belongs in the MacTech Spotlight, let us know! Send details to editorial@mactech.com

Advertiser/Product Index

Ad Index by Company for: MacTech June/2010

Ad Index by Product for: MacTech June/2010

THE MACTECH SPOTLIGHT

BOISY G. PITRE

http://www.tee-boy.com/

What is your company?

Tee-Boy. I own the company, and we are located in the heart of Cajun Country in Southwestern Louisiana.

What do you do?

My designated title is "Lead Developer" but I also run the company and handle all aspects of the business. We focus on custom programming on a consultancy basis for the Mac, iPhone, and iPad platforms, but also have a portfolio of products that we sell on our website.

How long have you been doing what you do?

Professionally, I've been a software engineer for 18 years. If you count my first exposure to computing and learning how to program, it's more like 28 years. More recently, I fell in love with the Mac back in 2002 and started learning Cocoa and Objective-C then; I've been doing Mac programming now for 8 years. It's by far the best platform that I've worked on.

What was your first computer?

My very first computer was a 16K Tandy Color Computer 2 (a.k.a. CoCo 2) from Radio Shack, followed shortly thereafter by a 128K CoCo 3. At first I used a cassette recorder to store and load programs until I could afford a disk drive unit. The CoCo came with a built-in BASIC interpreter, and I later learned 6809 assembly language, then moved onto the OS-9 (Microware, not Apple) operating system. Most people are surprised when I tell them that to this day I still have fun with the CoCo, and even run a retro-computing hardware/software business with a friend. See http://www.cloud9tech.com. Years later, my first Mac would be a 700MHz eMac G4. It still works today.

Are you Mac-only, or a multi-platform person?

I like working with Linux, and if it is absolutely necessary I can do Windows, but I would much rather work on Mac OS X. In fact, we're all Mac here at the office and also at the house. At home, we're huge advocates for the platform. My wife and I counted the number of folks that we've personally converted over to the Mac, and it's well beyond two-dozen and still going up.

What attracts you to working on the Mac?

Two things: simplicity and elegance. The Mac platform is frustration-free, beautiful to work in, and is a real pleasure to use. Apple has got it absolutely right in my opinion, and that is corroborated by the feedback that I get from others who have listened to my advice and purchased a Mac. I cannot tell you how many frustrating hours I have spent trying to get things done on other platforms, only to find the Mac does those same tasks effortlessly. Another benefit for me is that the Mac is built on such a robust operating system platform, BSD. It's almost a dichotomy that such a gorgeous and responsive user interface runs atop a powerful yet often-times misunderstood operating system platform, but Apple has the secret sauce that makes these two concepts gel so well. You cannot help but admire the engineering effort that went into integrating all of this.

What's the coolest thing about the Mac?

The coolest thing for me is that it "just works." It's really that simple.

What is the advice you'd give to someone trying to get into this line of work today?

First, I would say pay your dues by gaining some relevant programming experience. A formal education doesn't hurt either. If you're a prodigy, good for you, but a good grounding in theory and a lot of practice is a recipe for success. Second, always make your customer, client or employer's requirements a priority. Share their vision for their project and put your whole heart into making them happy with your work. Third, be passionate; it will show through your work and you will excel. Finally, be yourself. Don't be afraid to showcase your uniqueness and your personality. People want to work with other interesting people.

What's the coolest tech thing you've done using OS X?

There are several things, but my coolest achievement was getting my Mac to act as a disk, terminal, and MIDI server to my CoCo 3 using a product I designed and wrote called DriveWire. I developed the initial server software in Objective-C and Cocoa and now my Mac does my CoCo's bidding. I actually presented a poster at the WWDC07 Scientific Poster Development Session showcasing a compiler project that I

Continues on page 78

Mac shopping made easy.

Grab that to-do list, and prepare for some one-stop shopping at Smalldog.com!

Bundles simplify the buying process

Mac bundles (think Mac + RAM + AppleCare + external hard drive, etc.) not only include **everything you need,** but also **save you money.**

Visit » **Smalldog.com/specials**

Macs from under $500

We carry all **current Macs** as well as **used, refurbished and closeout models,** so there is a Mac for any budget.

Visit » **Smalldog.com/macs**

Free shipping over $200

It's true–we provide **free, same-day ground shipping** on every item over $200 every day.

Tax-free shopping

Purchases outside of Vermont are always shipped **tax-free.**

✓ 13" MacBook Pro + Chill Pill® mobile speakers

Small Dog Electronics

Always By Your Side

www.smalldog.com

800-511-MACS

Apple Specialist

Celebrating 15 Years • 3rd Largest Apple Specialist in New England • 5-Star Merchant Rating • Same-day shipping

Bundles

Macs

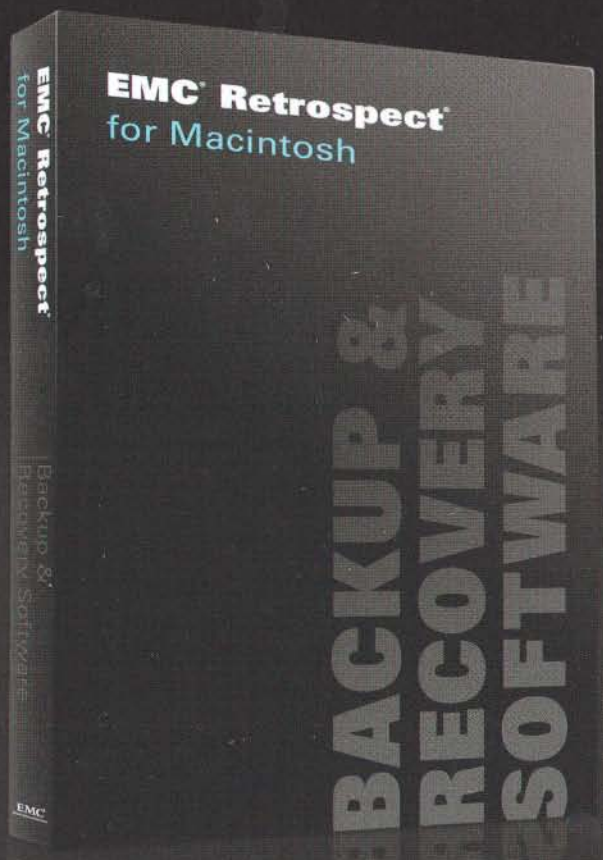
Free Shipping

Tax-Free

NEW!

EMC® Retrospect® 8

backup and recovery software for small and medium businesses



The most trusted name in Mac backup

All-new EMC Retrospect 8 for Macintosh provides the reliability, ease of use, power, and flexibility you need to protect critical data on Mac and Windows PCs and servers. EMC Retrospect includes a state-of-the-art Mac user interface and enterprise-level features — including remote management of one or more backup servers, disk-to-disk-to-*anything* backups, Xsan support and custom reporting — at a fraction of the cost of other products.

Download a free 45-day trial at **www.retrospect.com/wwdc**

EMC²
where information lives®